ABC INFORMATIQUE

N°64

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE

Les Commodore Plus/4 et 16

Logo : déplacements graphiques

Jouez au morpion

Macro-instructions sur Lotus 1-2-3

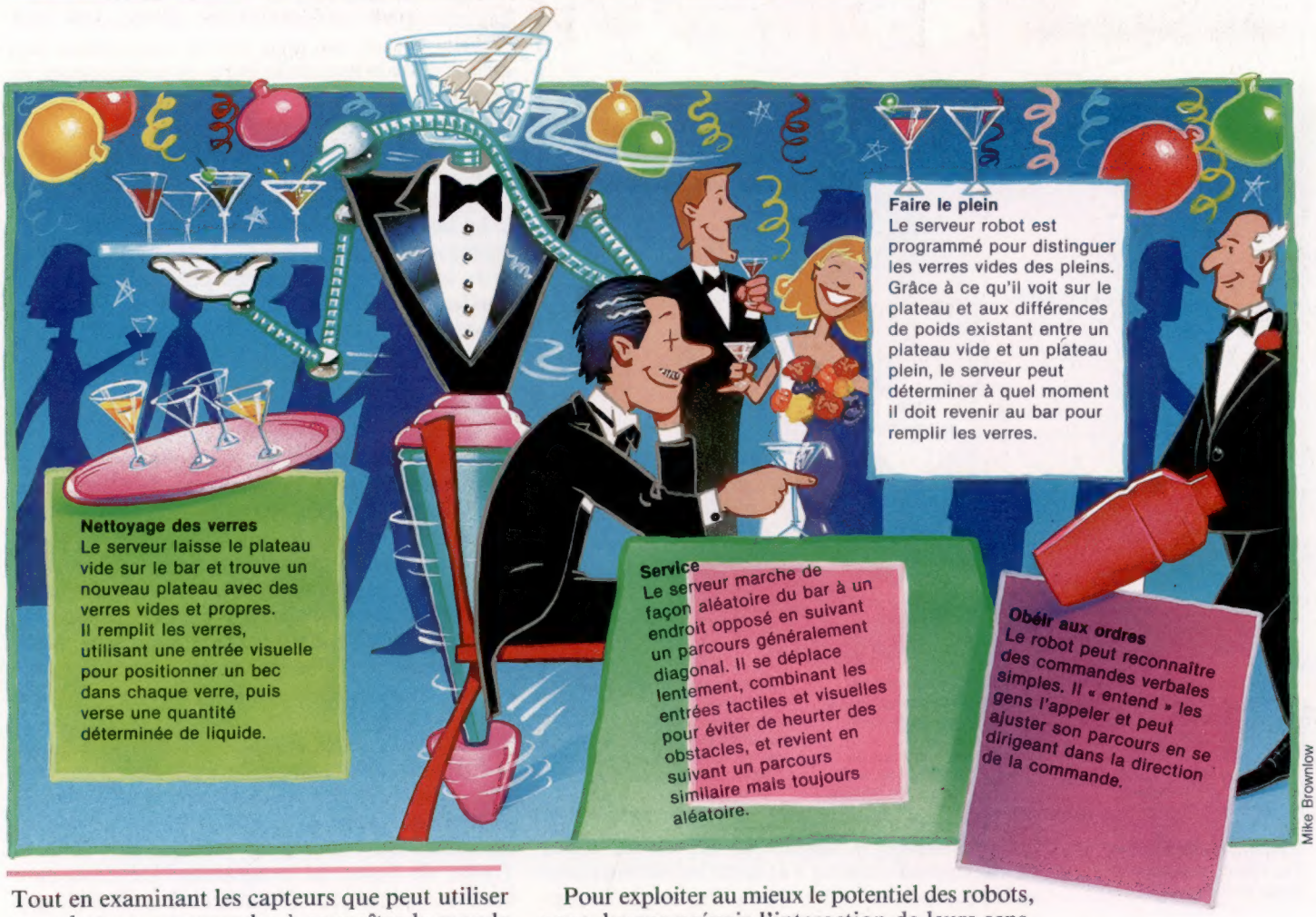
EDITIONS ATLAS

M 6062-64-12,50 F 90 FB - 3,80 FS - $ 1.95

Page manquante
(publicité et colophon)

La connaissance

Nous avons déjà expliqué de façon détaillée les différents « sens » qui contribuent à créer l'« intelligence » d'un robot. Examinons ici comment ces « sens » peuvent être combinés.



Mike Brownlow

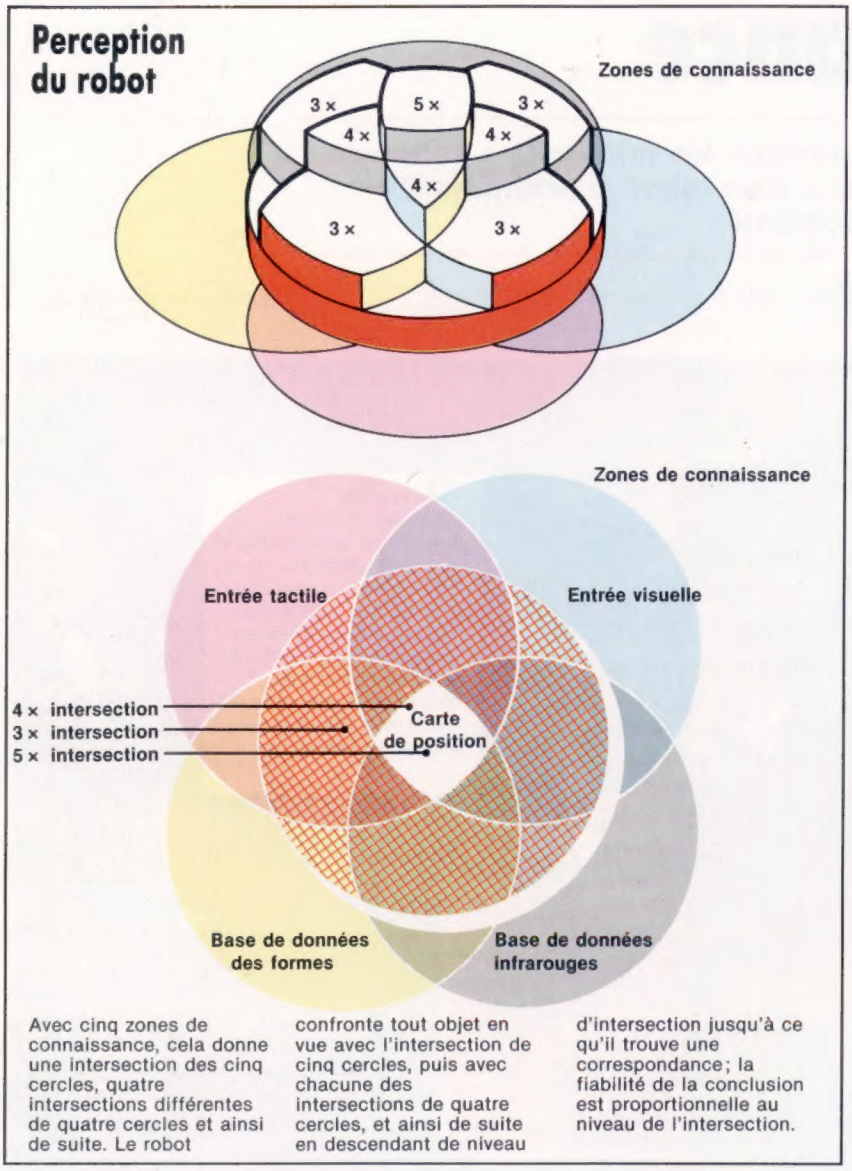
A votre service...
Nous avons créé un robot serveur imaginaire qui doit intégrer diverses données des capteurs pour remplir ses fonctions. Son plus grand problème réside dans la mobilité des clients, ce qui signifie qu'il doit constamment mettre à jour son modèle interne de l'espace.

Tout en examinant les capteurs que peut utiliser un robot pour apprendre à connaître le monde dans lequel il se déplace, nous avons étudié chaque type de capteur (lumière, son, toucher) de façon isolée. Cette approche est appropriée si le robot n'a qu'un seul capteur mais, en pratique, les bons robots en ont plusieurs. Pour comprendre son environnement, le robot doit être en mesure d'intégrer les données fournies par les capteurs en les confrontant entre elles ce qui lui permettra de composer un modèle complet du monde qui l'entoure.

Les êtres humains fonctionnent également de cette manière. Nos sens ne fonctionnent pas de façon isolée. Un bon exemple nous est donné par le cas des personnes aveugles de naissance et qui commencent à voir à la suite d'une intervention chirurgicale. La vitesse à laquelle ces patients utilisent pleinement leur vision surprend souvent. Cela s'explique par le fait que les aveugles ont une assez bonne connaissance du monde grâce au toucher et à l'ouïe.

Pour exploiter au mieux le potentiel des robots, nous devons prévoir l'interaction de leurs sens. Par exemple, un robot conçu pour saisir des objets peut être capable de le faire tout en étant « aveugle », mais il pourrait être bien plus efficace s'il était doté d'un système de vision, parce qu'il pourrait localiser les objets même s'ils étaient légèrement déplacés ou situés à un angle autre que ceux pour lesquels il avait été programmé. Pour ce faire, le robot doit entreprendre la représentation d'une construction de l'environnement à partir des informations fournies par ses capteurs et enregistrées dans sa mémoire. Il doit être capable de regarder un objet et de le reconnaître afin de positionner son bras et faire les calculs nécessaires pour saisir l'objet.

L'illustration la plus simple de ce modèle interne est donnée par le robot dans un labyrinthe qui utilise les capteurs pour trouver la position des parois. Il élabore une carte interne à deux dimensions tout en se déplaçant. Si nous étendons ce concept à un bras, la carte doit être tri-



Avec cinq zones de connaissance, cela donne une intersection des cinq cercles, quatre intersections différentes de quatre cercles et ainsi de suite. Le robot confronte tout objet en vue avec l'intersection de cinq cercles, puis avec chacune des intersections de quatre cercles, et ainsi de suite en descendant de niveau d'intersection jusqu'à ce qu'il trouve une correspondance; la fiabilité de la conclusion est proportionnelle au niveau de l'intersection.

Interface d'intersection
Les robots disposent généralement de plusieurs sources de connaissance : il peut s'agir de bases de données préprogrammées (silhouettes d'objets communes et signatures infrarouges), de bases de données expérimentales (comme la carte de l'environnement du robot) et de canaux d'entrée de capteurs (tactiles ou visuels). Idéalement — lorsque le robot « connaît » sa position et « comprend » son environnement — l'intersection de zones de connaissance lui permet de reconnaître tout objet placé devant lui. (Cl. Kevin Jones.)

dimensionnelle. L'apport de la vue au robot donne immédiatement à la carte les informations sur la couleur, la brillance et la forme que les capteurs tactiles ne pouvaient seuls détecter. Avec une certaine reconnaissance de la parole, le robot ajoute une information verbale à son modèle du monde.

Le problème qui est posé aux concepteurs de robot réside dans le fait que le monde n'est pas statique mais qu'il change continuellement. Le robot doit donc être en mesure d'en tenir compte.

Si nous prenons un robot qui est programmé pour effectuer certaines tâches simples comme empiler des briques, l'étendue de ce problème devient claire. Si les briques sont de dimensions inégales, elles doivent être placées les unes au-dessus des autres avec beaucoup de précautions; car si le centre de gravité se déplace à l'extérieur de la zone de base, la pile entière s'écroulera. mais quelle connaissance le robot a-t-il des lois de la gravité? Et si la pile s'effondre, comprendra-t-il ce qui est arrivé et prendra-t-il les mesures appropriées?

Solution du problème

Il existe deux approches principales pour résoudre ce problème. La première consiste à programmer le robot avec des données qui comportent une description des interventions prescrites pour chaque éventualité. Cela restreint la compréhension du robot à certaines tâches clairement définies dès le départ.

Le robot empileur de briques sera donc programmé avec des instructions qui assurent que chaque brique est bien placée exactement au-dessus de la brique inférieure, en positionnant le centre de gravité de l'une directement au-dessus du centre de gravité de l'autre.

La seconde approche est défendue par ceux qui estiment que la seule manière de permettre à un robot de comprendre son environnement est de le lui faire découvrir. C'est un domaine de l'informatique que l'on nomme « heuristique ». Avec cette approche, le robot est programmé en vue d'effectuer une tâche particulière et profite d'une réaction, soit en provenance d'une personne, ou de son propre capteur qui le renseigne sur la qualité de son intervention. A l'aide de ces retours d'informations, le robot modifiera son propre programme interne — son propre modèle du monde — afin d'améliorer ses performances et de constituer une « bibliothèque de référence » qui l'aidera dans de futures tâches. Le robot du labyrinthe agit de la même manière lorsqu'il essaie de trouver la meilleure sortie. A l'aide de ses capteurs, il peut détecter un cul-de-sac et prendre les mesures qui s'imposent pour revenir sur ses pas et pour essayer un autre parcours. Malheureusement, il n'existe pas un programme unique qui peut servir à enseigner toutes ces tâches à un robot.

Dès que le robot a appris la leçon nécessaire, il doit la stocker dans un programme d'ordinateur. Cette opération se nomme « représentation de la connaissance ». Traditionnellement, la connaissance du robot peut être stockée sous la forme de lignes de programme informatique. Mais les techniques d'intelligence artificielle ont conduit à d'autres approches. De nombreuses techniques sont utilisées, mais les plus répandues s'appellent *règles de production, réseaux sémantiques* et *cadres.*

Les règles de production se présentent comme des constructions SI...ALORS (IF...THEN) et ne sont que de simples formulations de faits. SI il y a un mur de briques face à vous, ALORS vous ne pouvez avancer. Il peut y avoir une séquence complète de règles de ce type — elles ont l'avantage d'être faciles à écrire et faciles à comprendre pour celui qui écrit le programme. Le problème est que le robot doit lui aussi les comprendre, et ce, pendant le déroulement d'une intervention. Les programmes qui utilisent les règles de production peuvent être écrits dans un langage conventionnel, comme BASIC, mais sont plus fréquemment écrits dans des langages *déclaratifs,* comme PROLOG, qui est particulièrement adapté à ce type de connaissance. Cela s'explique par le fait que, contrairement aux langages traditionnels, les

langages déclaratifs n'exécutent pas leurs instructions une à la fois.

En fait, le programme recherche toujours un ensemble donné de circonstances auxquelles une ou plusieurs règles pourraient s'appliquer. Lorsque tel est le cas, cette règle particulière est exécutée, ce qui permet d'appeler l'exécution d'autres règles.

Les réseaux sémantiques sont une forme de structures graphiques servant à représenter la connaissance. Il est possible de les comparer à un simple réseau de relations existant entre divers éléments de connaissance. Ils sont appelés ainsi pour la simple raison que les liaisons individuelles ont une certaine signification en tant que telles. Au lieu d'être uniquement en présence d'un arc reliant deux nœuds, cet arc peut indiquer l'existence d'un type spécial de relation entre les deux nœuds. Ainsi, un nœud nommé « table » peut être relié à un nœud nommé « meuble ». Dans cet exemple, la relation est la suivante : la table est un type de meuble.

Ce genre de représentation de la connaissance peut être programmé dans un langage conventionnel. Vous pouvez essayer d'utiliser BASIC et représenter les divers nœuds et liaisons par des variables de chaîne. Mais il est plus fréquent d'utiliser des langages d'intelligence artificielle — comme LISP — qui sont plus faciles pour exprimer ces relations complexes.

Questionnaire

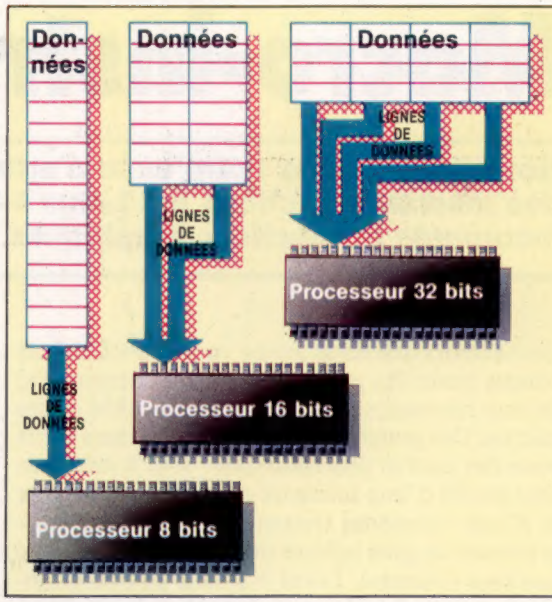
Les cadres ressemblent un peu à un questionnaire vierge spécialement conçu pour chaque type de situation pouvant être rencontrée par un robot. Cette notion est très facile à comprendre et peut être programmée sans difficultés en BASIC à l'aide de simples tableaux bidimensionnels — une dimension pour la « question » et une autre pour les « réponses ».

Avec cette approche, on ne considère pas que le robot a une entière connaissance d'une situation tant qu'il n'a pas rempli tous les éléments du questionnaire. Ce n'est qu'à ce moment qu'il peut entreprendre une action. Cette méthode peut être raffinée en offrant au robot un large choix de cadres différents. L'une des tâches du robot est alors de choisir le cadre approprié à une situation donnée.

Un aspect important de la représentation de la connaissance que nous avons traité brièvement est le rôle joué par les langages de programmation eux-mêmes.

LISP, par exemple, étant un langage de traitement de liste, est particulièrement approprié à ce type d'approche pour le stockage et la récupération de données. Le choix du langage pourra donc faciliter la représentation de certains types de connaissance.

Toutes ces méthodes font l'objet d'importantes recherches dans les départements d'informatique des universités de nombreux pays. Les éléments clés des travaux sont l'utilisation de capteurs, la réaction, l'apprentissage et la représentation de la connaissance.



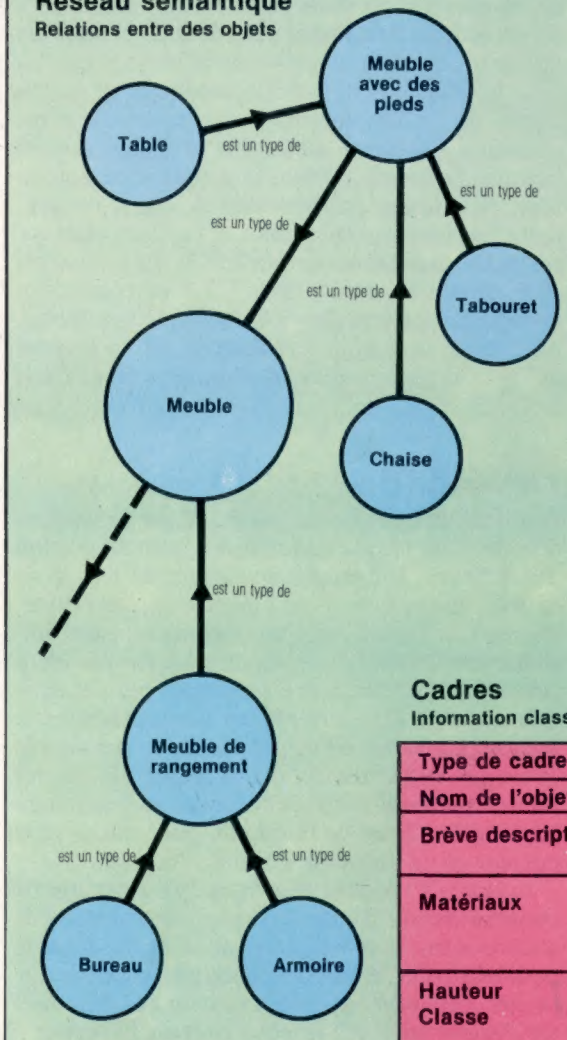
Lignes parallèles

Les données fournies par les capteurs doivent être traitées suffisamment rapidement. La longueur de la file d'attente des entrées est déterminée par la complexité de l'algorithme qui les interprète et par la vitesse de traitement. Le volume des données générées par un robot perfectionné peut être telle qu'un processeur à 8 bits comme le Z80 ou le 6502 ne sera pas assez rapide, ce qui entraîne la formation de longues files d'attente de données. La solution consiste à utiliser des processeurs 16 ou 32 bits.

Les chemins de la connaissance

Réseau sémantique

Relations entre des objets



La connaissance peut être définie comme étant une information placée dans un contexte précis. Les ordinateurs sont construits pour stocker de l'information, mais ne sont pas spécialement équipés pour gérer les relations qui existent entre les données. Il est donc nécessaire de concevoir des méthodes appropriées pour les représenter. Parmi celles-ci, citons les réseaux sémantiques qui peuvent être stockés comme une liste enchaînée; les cadres, qui ne sont que des tableaux bidimensionnels; les règles de production, qui sont des listes d'informations et d'opérateurs logiques. Aucune de ces méthodes n'est idéale pour représenter la connaissance, et elles sont souvent combinées. Ici, nous utilisons un réseau sémantique pour représenter une connaissance détaillée.

Cadres

Information classée

Type de cadre	1
Nom de l'objet	Tabouret
Brève description	3 pieds Siège
Matériaux	Bois Métal Plastique
Hauteur **Classe**	0,5-1 m Meuble avec pieds Sièges

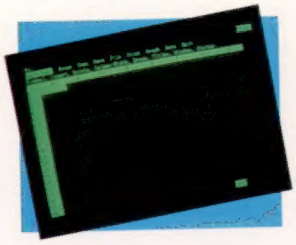
Règles de production

Listes boléennes

SI tabouret ALORS (3 pieds **ET** siège) **ET** (bois **OU** métal **OU** plastique)

Macro-instructions

Nous poursuivons notre série d'articles sur les tableurs par l'étude des macro-instructions sur Lotus 1-2-3, tableur avec base de données incorporée et fonctions graphiques.



L'écran Lotus
Le menu principal s'obtient avec Lotus en frappant la touche /, comme pour VisiCalc.

Macro-économie
Voici notre macro d'appellation d'une zone, telle qu'elle figure au tableau. Les macro-instructions peuvent s'appliquer si nécessaire à une colonne.

Les tableurs que nous avons vu jusqu'ici ont été conçus pour des micro-ordinateurs personnels. Ils sont nécessairement limités par la RAM de ces micros. Des programmes semblables, mais écrits pour des micros professionnels, sont à même de tirer profit d'une mémoire résidente importante et d'une vitesse de traitement élevée. Les programmes les plus novateurs supposent donc une machine coûteuse. Lotus 1-2-3 est un bon exemple de tableur avec base de données et logiciel graphique intégrés.

Par sa conception, Lotus 1-2-3 suppose, pour son exploitation, une quantité importante de mémoire résidant dans l'ordinateur. Outre suffisamment de RAM pour gérer le code de ses trois applications, il faut encore de la place mémoire pour le tableau dont la taille maximale est de 256 lignes de 1 028 colonnes. La mémoire n'est allouée au tableau qu'au fur et à mesure des besoins. Et pourtant, dans la première version du logiciel, il fallait un minimum de 128 K uniquement pour son exploitation. Les versions ultérieures supposant au minimum 256 K. Le coût d'un programme tel que Lotus 1-2-3 et celui d'un système pour l'exploiter sont extrêmement élevés, mais le résultat pour l'utilisateur est un logiciel aux ressources très grandes. Comment fonctionne Lotus 1-2-3?

Puissance

Une fois chargé, Lotus 1-2-3 affiche le système d'accès : un jeu de commandes pour la gestion des données. En plaçant le curseur en regard de la première option, et en la validant (par retour-chariot), le tableau se charge en mémoire et initialise son écran. Les lignes du tableau sont identifiées par des lettres, et les colonnes par des chiffres. Lotus 1-2-3 est piloté par l'intermédiaire de menus. Le menu principal s'affiche sur simple pression de la touche «/». A partir de là, les options du menu sont sélectionnées soit en tapant la première lettre de la commande, soit avec le curseur et en validant l'option désignée.

Lotus 1-2-3 a un si grand nombre d'options de commandes qu'il comporte plusieurs niveaux de sous-menus. Si cela signifie pour l'utilisateur la possibilité d'effectuer des centaines de tâches distinctes, cela veut également dire un nombre élevé de frappes successives pour obtenir l'exécution d'un travail déterminé (voir le schéma). Pour s'en convaincre, il suffit de prendre un exemple : Lotus 1-2-3 vous permet de donner un nom à une position ou à un ensemble de positions du tableau, par une étiquette. Lorsque vous désirez agir sur la région ainsi nommée pour la faire figurer dans une formule mathématique par exemple, vous utilisez ce nom en tant que référence à cette position, de la façon suivante :

A3 – B3 = C3	VENTES – COÛT = PROFIT
références des positions	références des noms

Nommer ainsi des zones de positions simplifie et accélère la recherche sur les grands tableaux. Pour donner un nom à un ensemble de quatre positions, il faut suivre la démarche suivante :

/ –, pour afficher le menu.
É(cart), pour spécifier la zone concernée (et dire qu'il ne s'agit pas de la totalité du tableau).
N(om), pour l'ensemble des positions qui doit porter un nom distinctif.
C(réer), pour sauvegarder le nouveau nom.
Tapez le NOM à attribuer, «DÉBUT» par exemple; faites un retour-chariot pour valider la première position de la zone par l'intermédiaire de la position active; déplacez le curseur de quatre positions sur la droite. Tapez sur retour-chariot pour valider la fin de la zone de la même manière.

Le tout a pris dix frappes successives, plus celle du nom.

Pour réduire cette longue frappe, Lotus 1-2-3 a prévu les macro-instructions. Il s'agit de simples programmes écrits dans le système d'exploitation du logiciel. Les macro-instructions sont créées en sauvegardant les frappes successives sur une petite partie du tableur, en donnant un nom à la zone, et en assignant ce nom à une touche spéciale du clavier. La frappe de cette touche en conjonction avec une touche unique de fonction (valable pour toutes les macrocommandes), appelée ALT sur l'IBM PC et ses compatibles, exécute la suite des commandes élémentaires correspondant aux frappes sauvegardées.

Pour rendre automatique l'appellation de positions, commençons par allouer une zone du tableur à la macro-instruction. Cette zone qui loge la macrocommande doit être choisie avec soin. D'abord, elle doit être située dans un endroit très sûr du tableau que les données ne pourront pas atteindre. Ensuite, Lotus n'alloue de l'espace mémoire qu'aux positions activées. Une position est dite activée lorsqu'elle a été pointée par le curseur. Cela signifie que les positions vides, comprises entre deux positions de données ayant été activées, le seront également et occuperont de la place mémoire. Ainsi, lorsqu'une macro-instruction est placée loin vers la droite du tableau, plusieurs kilo-octets peuvent

être perdus à loger des positions vides. Il est préférable de mettre les macro-instructions vers la gauche du tableau, aux colonnes A, B et C par exemple. Si toutes les formules mathématiques sont écrites pour s'effectuer de gauche à droite, les zones réservées aux macro-instructions resteront toujours à l'abri.

Nous établirons notre macro d'« appellation » dans la colonne A. Lorsque le nombre de frappes devient trop important pour figurer sur une seule position, les positions voisines sont occupées tour à tour. Après avoir pointé le curseur sur A1, nous tapons les frappes voulues :

/ É(cart) N(om) C(réation)

Lotus 1-2-3 marque alors une pause avant d'enregistrer le nom choisi. Il est toujours possible de demander une pause en tapant un point d'interrogation entre parenthèse (?). Le logiciel attend alors la frappe de RC (Retour-Chariot) pour reprendre. La syntaxe () est utilisée systématiquement pour demander une action ne correspondant à aucune commande au clavier. Les déplacements du curseur correspondent ici à ce cas et supposent que l'on écrive la direction souhaitée entre parenthèse. Les RC sont indiqués par le signe « tilde » : ~. La définition de la macro-instruction se poursuit donc :

' / E N C (?) ~(droite) (droite) (droite) (droite)

Nous avons alors tapé l'identification d'une région pour recevoir la macro. Il s'agit du corps de la macro.

L'étape suivante consiste à donner un nom à cette région, par une lettre au clavier représentant le nom par exemple (N pour Nom). Malheureusement, nous nous trouvons devant un cercle vicieux. Nous devons saisir toutes les frappes qui viennent d'être tapées pour la macro de définition des limites de positions, parce que celle-là n'a pas encore de nom et n'est pas sauvegardée ! Nous positionnons le curseur en A1 et tapons :

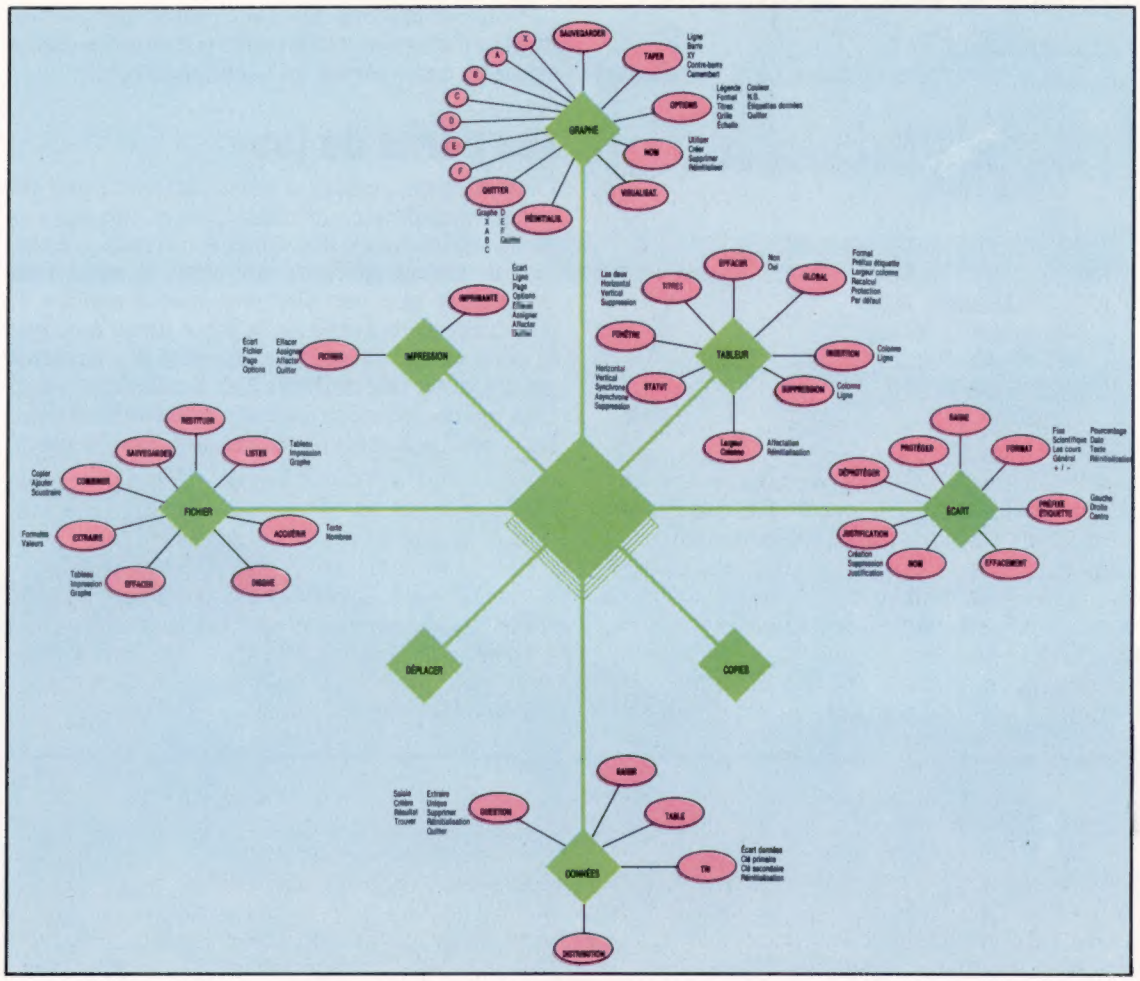
'/ E N C \ N RC RC

Le signe \ indique que la touche ALT doit être frappée simultanément avec la touche suivante. \ N signifie donc ALT N pour Lotus 1-2-3. Maintenant que la région est nommée, la frappe ALT et N génère automatiquement la séquence voulue. Il est alors facile de nommer une région pour une macro-instruction en tapant :

ALT-N NOM RC

ce qui représente un progrès immense par rapport au premier processus.

Cet exemple, bien qu'utile, n'est qu'un petit échantillon des potentialités des macro-instructions. Il n'existe théoriquement pas de limites au nombre de commandes élémentaires (les frappes) que peut comporter une macro-instruction, ni de restrictions au type d'opération qu'il est ainsi possible de rendre automatique.



Arbre de commande de Lotus
Lotus comporte plusieurs niveaux de menus, comme on peut le voir ici. Ces commandes sont incorporées. D'autres peuvent être définies par l'utilisateur par le biais de macro-instructions au clavier.
(Cl. Lotus Development Corp.)

Poursuite périlleuse

Nous avons vu comment définir avec LOGO les procédures de base d'un jeu d'aventures. Nous traitons ici des procédures de déplacement d'objets graphiques entre les pièces.

La structure de base du jeu étant définie, nous devons passer aux procédures pour déplacer les objets d'une pièce à l'autre. Nous créons quatre directions pour les déplacements : vers le nord, l'est, l'ouest et le sud.

```
POUR N
   DÉPLACE «N :LISTE.SORTIES
FIN
POUR S
   DÉPLACE «S :LISTE.SORTIES
FIN
POUR E
   DÉPLACE «E :LISTE.SORTIES
POUR O
   DÉPLACE «O :LISTE.SORTIES
```

Une procédure appelée DÉPLACE vérifie qu'un déplacement dans la direction demandée est possible, et laisse à une autre le soin d'exécuter le déplacement lui-même — DÉPLACEMENT1.

```
POUR DÉPLACE :DIR :LISTE
   SI VIDE? :LISTE ALORS AFFICHE [VOUS NE POUVEZ ALLER
     PAR LÀ] STOP
   FAIT «SORTIE PREMIER :LISTE
   SI :DIR = PREMIER:SORTIE ALORS DÉPLACEMENT1
     DERNIER :SORTIE
   STOP
   DÉPLACE :DIR SAUFDERNIER :LISTE
FIN
POUR DÉPLACEMENT1 :NO
   FAIT :NOM.PIÈCE DÉTAILS.ICI
   FAIT «ICI :NO
   ASSIGNE.VARIABLES
   REGARDE
FIN
```

DÉPLACEMENT1 accepte en entrée un numéro de pièce. Elle commence par réagencer une liste à partir de ses divers composants, et l'assigne au nom de la pièce (il y a peut-être eu des changements après le passage de l'aventurier). Elle donne ensuite à ICI la valeur de la nouvelle pièce désignée et redistribue l'affectation des différentes listes. Voici la procédure utilisée :

```
POUR ICI.DÉTAILS
   RÉSULTAT (LISTE :DESCRIPTION :CONTENU
   :LISTE.SORTIES
FIN
```

Cette dernière utilise la primitive LISTE qui recense les entrées. La différence entre LISTE et PHRASE peut se voir sur un exemple :

```
LISTE [A] [B] [C] donne [[A] [B] [C]]
PHRASE [A] [B] [C] donne [A B C]
```

Nous utiliserons ici LISTE plutôt que PHRASE dans la mesure où nous voulons garder les divers éléments sous forme de sous-ensembles.

Les périls du jeu

Généralement, un jeu d'aventures comporte un certain nombre de « périls » à éviter, tels des serpents venimeux ou des sables mouvants... Lorsque le joueur affronte un péril, il nous faut déclencher une certaine séquence d'actions et empêcher toute sortie de la pièce jusqu'à ce que le péril ait été vaincu. La manière d'y parvenir est d'ajouter une nouvelle liste à celle de la pièce, liste qui contienne le nom des procédures spéciales « péril » à exécuter en entrant dans la pièce.

Nous pouvons ainsi définir PIÈCE.2 comme [[[VOUS ÊTES DANS UNE CAVE NOIRE ET HUMIDE] [LA LUMIÈRE EST EN FACE DE VOUS] [BOÎTE]] [[N5] [E6]] [SERPENT]] où SERPENT est un « péril ». Après avoir ajouté cela à notre liste, il nous faut modifier la procédure REGARDE :

```
POUR REGARDE
   AFFICHE :DESCRIPTION
   AFFICHE «
   AFFICHE [VOUS VOYEZ:]
   SI VIDE? :CONTENU ALORS AFFICHE [RIEN DE SPÉCIAL]
     SINON AFFICHE :CONTENU
   AFFICHE «
   AFFICHE [VOUS POUVEZ Y ALLER :]
   AFFICHE.SORTIES :LISTE.SORTIES
   AFFICHE «
   SI PÉRIL? ALORS EXÉCUTE :PÉRILS
FIN
```

RUN est une primitive LOGO très puissante. Il reçoit en entrée une liste et traite les procédures qu'elle contient. Ici, [SERPENT] pourra être assigné à PÉRILS, de sorte que EXÉCUTE :PÉRILS exécute SERPENT.

```
POUR PÉRIL?
  SI PÉRIL? :PÉRILS ALORS RÉSULTAT «FAUX
  RÉSULTAT «VRAI
FIN
```

D'autres procédures sont à modifier pour tenir compte des périls :

```
POUR ASSIGNE.VARIABLES
   FAIT «NOM.PIÈCE MOT «PIÈCE. :ICI
   FAIT «PIÈCE CHOSE :NOM.PIÈCE
   FAIT «DESCRIPTION DESCRIPTION :PIÈCE
   FAIT «CONTENU CONTENU :PIÈCE
   FAIT «LISTE.SORTIES LISTE.SORTIES :PIÈCE
   FAIT «PÉRILS PÉRILS :PIÈCE
FIN
POUR PÉRILS :PIÈCE
   RÉSULTAT ÉLÉMENT 4 :PIÈCE
FIN
POUR ICI.DÉTAILS
   RÉSULTAT (LISTE :DESCRIPTION :CONTENU
     :LISTE.SORTIES :PÉRILS)
FIN
POUR DÉPLACE :DIR :LISTE
   SI PÉRIL? ALORS AFFICHE [VOUS NE POUVEZ
     ALLER PAR LÀ] STOP
   SI VIDE? :LISTE ALORS [VOUS NE POUVEZ
     ALLER PAR LÀ] STOP
   FAIT «SORTIE PREMIER :LISTE
   SI :DIR = PREMIER :SORTIE ALORS DÉPLACEMENT1
     DERNIER :SORTIE
   STOP
   DÉPLACE :DIR SAUFPREMIER :LISTE
FIN
```

DÉPLACE empêche tout déplacement jusqu'à ce que PÉRILS reçoive la valeur []. En mettant en place les périls de la sorte, il est possible d'utiliser le même péril pour plusieurs pièces, et de le déplacer d'une pièce à l'autre en modifiant simplement la description de la pièce.

Nous pouvons maintenant utiliser les procédures que nous avons développées jusqu'ici pour élaborer un jeu complet d'aventures appelé Les Reliques de Zoltoth. Dans ce jeu, l'aventurier est à la recherche du sceptre de Gilgesh qui a été volé par les grands prêtres de Zoltoth et enfoui dans la crypte du temple. Au début du jeu, l'aventurier est à l'entrée de la crypte. Si vous écrivez votre propre version du jeu, commencez par concevoir un scénario pour une partie gagnante; écrivez-le et structurez tout le jeu sur cette base. Nous ne le donnons pas ici, ce qui vous laisse le champ libre.

L'étape suivante est de diviser le jeu en termes de « pièces » : localisations dont les contenus et situations sont interdépendants. Ce schéma d'un monde fantastique sert à définir les positions, transposées dans le programme, qui donnent les sorties possibles pour chaque localisation. Les aventuriers devront dresser une carte au fur et à mesure de leur progression.

Il nous faut maintenant définir le vocabulaire du jeu : quels mots, dits par l'aventurier, le jeu sera-t-il en mesure de comprendre? Voici le langage existant entre le jeu et l'aventurier :

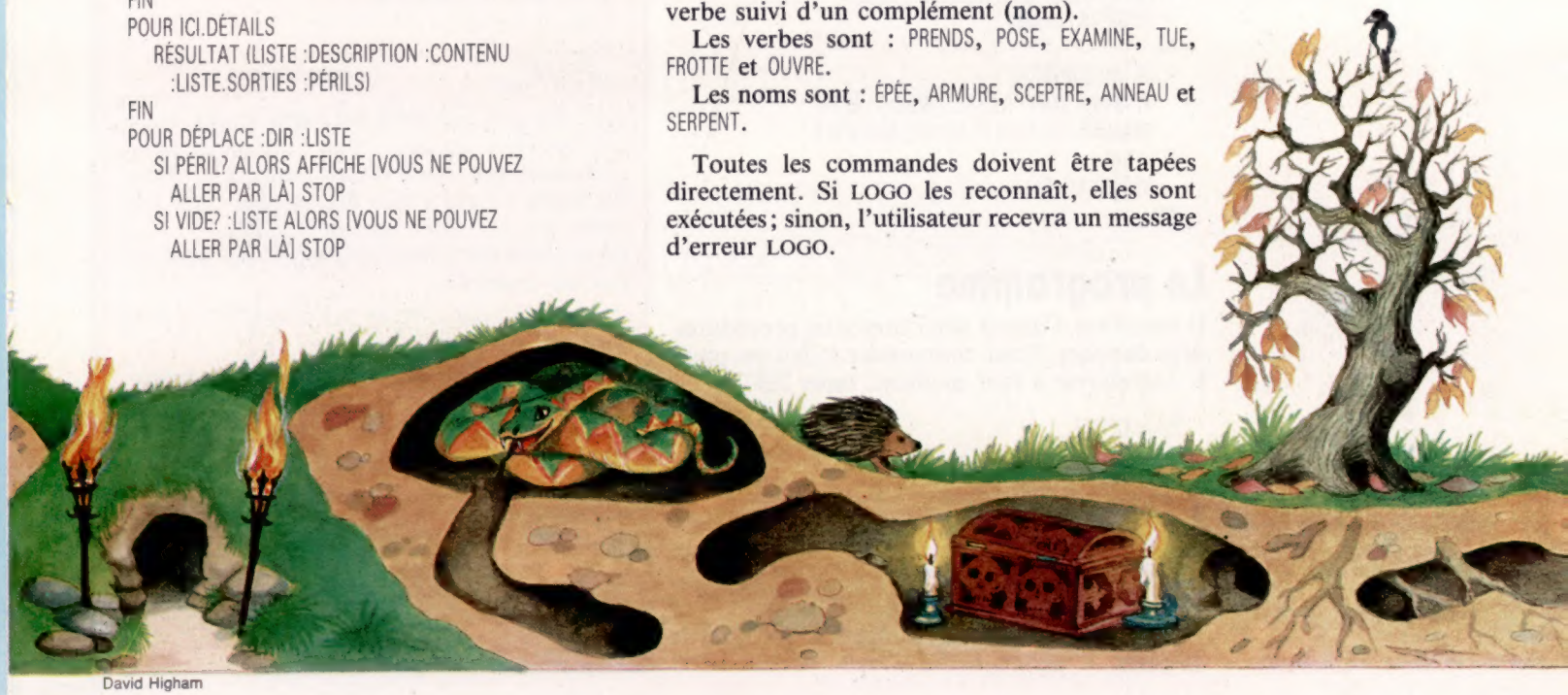
1. Sept mots simples de commande : DÉBUT, REGARDE, N, S, E, O, et INVENTAIRE (tous déjà décrits).

2. Mots de commande doubles composés d'un verbe suivi d'un complément (nom).

Les verbes sont : PRENDS, POSE, EXAMINE, TUE, FROTTE et OUVRE.

Les noms sont : ÉPÉE, ARMURE, SCEPTRE, ANNEAU et SERPENT.

Toutes les commandes doivent être tapées directement. Si LOGO les reconnaît, elles sont exécutées; sinon, l'utilisateur recevra un message d'erreur LOGO.



David Higham

Il serait néanmoins préférable que soit communiqué un message approprié, du genre « Je ne connais pas ce nom », plutôt que le message standard d'erreur LOGO. A cette fin, il nous faut une boucle externe qui prend les données, en vérifie la validité et les exécute. Voici une des manières de la réaliser pour ce qui concerne le vocabulaire défini jusqu'ici :

```
POUR DÉBUT
   FAIT «ICI 1
   FAIT «INVENTAIRE [ ]
   POSITIONNE.PIÈCES
   ASSIGNE.VARIABLES
   REGARDE
   JEU
FIN
POUR JEU
   AFFICHE 1 «COMMANDE :
   FAIT «ENTRÉE ORDRE
   SI VALIDE? :ENTRÉE EXÉCUTE :ENTRÉE
   SINON AFFICHE [JE NE COMPRENDS PAS]
   JEU
FIN
POUR VALIDE? :COM
   SI ( ( COMPTE :COM) = 1 ALORS RÉSULTAT
   VAL1? :COM
   SI ( ( COMPTE :COM ) = 2 ALORS RÉSULTAT
   VAL2? :COM
   RÉSULTAT «FAUX
FIN
POUR VAL1? :COM
   SI APPARTIENT? PREMIER :COM [INV O E S N
   REGARDE DÉBUT] RÉSULTAT «VRAI
   RÉSULTAT «FAUX
FIN
POUR VAL2? :COM
   SI TOUS VALEURVERBE? PREMIER :COM VALEURNOM?
   DERNIER :COM RÉSULTAT «VRAI RÉSULTAT « FAUX
POUR VALEURNOM? :NOM
   SI APPARTIENT :NOM [ÉPÉE ARMURE SCEPTRE
   ANNEAU SERPENT] RÉSULTAT «VRAI
   RÉSULTAT «FAUX
FIN
POUR VALEURVERBE
   SI APPARTIENT? :VERBE [PRENDS LÂCHE
   EXAMINE TUE FROTTE OUVRE] RÉSULTAT
   «VRAI
   RÉSULTAT «FAUX
FIN
```

Le programme

Il nous faut d'abord saisir toutes les procédures déjà données. Pour commencer le jeu ou pour le redémarrer à tout moment, taper DÉBUT.

```
POUR DÉBUT
   FAIT «ICI 1
   FAIT INVENTAIRE [ ]
   POSITIONNE.PIÈCES
   ASSIGNE.VARIABLES
   REGARDE
FIN
```

POSITIONNE.PIÈCES localise les pièces en fonction de la carte.

```
POUR POSITIONNE.PIÈCES
   FAIT «PIÈCE.1 [VOUS ÊTES SUR LE SEUIL]
   [D'UNE CAVE]] [ ] [[E2]] [ ] ]
   FAIT «PIÈCE.2 [[[VOUS ÊTES DANS UNE CAVE NOIRE
   HUMIDE]] [ ] [[S3] [E4] [01]] [ ] ]
   FAIT «PIÈCE.3 [[[VOUS ÊTES DANS UNE CAVE NOIRE
   ET HUMIDE]] [ ] [[N2]] [E5]] [ ] ]
   FAIT PIÈCE.4 [[[VOUS ÊTES DANS UNE GRANDE PIÈCE
   SOUTERRAINE]] [ ] [[N6]] [S5]
   [02]] [SERPENT.ATTAQUE]]
   FAIT «PIÈCE.5 [[[VOUS ÊTES DANS UNE CAVE NOIRE
   ET HUMIDE]] [ÉPÉE] [[N4] [03]] [ ] ]
   FAIT PIÈCE.6 [[[VOUS ÊTES DANS LA CRYPTE SACRÉE DES
   RELIQUES] [DANS UNE NICHE DU MUR NORD] [EST UN
   AUTEL]] [ ] [[N7] [S4] [E8]] [PORTE]]
   FAIT «PIÈCE.7 [[[VOUS ÊTES DEVANT]
   [L'AUTEL DE ZOLTOTH LE DORÉ] [AU-DESSUS DE L'AUTEL
   EST ÉCRIT:] [«QU'AUCUN VIL MÉTAL N'APPROCHE]]
   [L'ANNEAU] [[S6]] [ ] ]
   FAIT «PIÈCE.8 [[[VOUS ÊTES DANS UNE CAVE NOIRE
   ET HUMIDE]] [ ] [[S10] [E9] [06]] [SERPENT.ATTAQUE]
   FAIT «PIÈCE.9 [[[VOUS ÊTES DANS UNE CAVE NOIRE
   ET HUMIDE]] [ARMURE] [[S11] [08]] [ ] ]
   FAIT «PIÈCE.10 [[[VOUS ÊTES DANS UNE CAVE NOIRE
   ET HUMIDE]] [ ] [[N8] [E11]] [ ] ]
   FAIT «PIÈCE.11 [[[VOUS ÊTES DANS LA SACRISTIE]
   [DU PRÊTRE DE ZOLTOTH LE DORÉ]] [SCEPTRE] [[N9] [010]]
   [ ] ]
FIN
```

Variantes de logo

Certaines versions MIT LOGO n'ont pas VIDE?, ÉLÉMENT, COMPTE ou APPARTIENT?. Nous avons donné précédemment des définitions pour ces procédures. Pour toutes les versions LCSI, vous utiliserez :

```
VIDEP pour VIDE?
LISTEP pour LISTE?
APPARTIENTP pour APPARTIENT?
TAPE pour AFFICHE1
ET pour TOUS
OU pour UNSEUL
```

EGALP est une primitive qui teste si ses deux entrées sont égales. Vous l'utiliserez pour comparer des listes et des mots, à la place du signe « égal » (qui est valable pour les listes sur certaines versions LCSI).
La syntaxe de SI dans LOGO LCSI est illustrée par cet exemple :

```
SI VIDEP : CONTENU [AFFICHE [RIEN DE SPÉCIAL]] [AFFICHE
:CONTENU]
```

La première liste après l'expression conditionnelle est exécutée quand la condition est vraie; et la seconde, lorsqu'elle est fausse. Sur LOGO Atari utilisez PH pour PHRASE, OR pour ORDRE et notez qu'ÉLÉMENT n'est pas implémenté.

Seize sur vingt?

CBM vient de lancer deux micro-ordinateurs destinés à l'usage personnel et à la petite gestion : le Plus/4 et le Commodore 16, moins onéreux mais dont la mémoire est plus réduite.

Le Commodore 16, de toute évidence, est prévu pour remplacer le Vic-20, qui ne disposait guère que de 3,5 K de RAM. Mais son successeur, comme le Plus/4, se voit enfin pourvu d'un BASIC puissant : commandes à l'unité de disquettes (un peu comme sur le CBM 8296), aides à la programmation, graphismes et sons... On notera par ailleurs une certaine évolution vers la programmation structurée, avec des constructions du type DO...WHILE et LOOP...UNTIL...EXIT.

Toutefois, à la différence du Plus/4, dont le clavier et le boîtier sont radicalement nouveaux, le 16 a la même apparence extérieure que le Vic-20 et le 64, bien que la disposition des touches et les couleurs (désormais gris clair et gris anthracite) aient été modifiées. Le clavier est bien conçu et il présente une allure et un toucher vraiment professionnels.

Une innovation intéressante, l'apparition d'une touche HELP. Elle permet, en cas d'affichage d'un message d'erreur à l'écran, de visionner la ligne incriminée, tout en illuminant la partie incorrecte. Si, cependant, la ligne contient plusieurs instructions, c'est tout ce qui est compris entre l'erreur elle-même et la fin de la ligne qui sera mis en valeur de cette façon — il aurait quand même été plus judicieux de ne faire clignoter que l'erreur elle-même.

Des points faibles

Un des aspects les plus ennuyeux du 16 et du Plus/4 est qu'aucun d'eux n'est compatible avec le Vic-20 ou le 64. Le matériel lui-même a été modifié : c'est ainsi que la prise cassette et la prise manche à balai ne sont plus les mêmes, car la seconde n'emploie plus, désormais, le traditionnel connecteur à neuf broches de type D. Il faut cependant noter que les interfaces destinées au lecteur de disquettes et au moniteur sont identiques à celles du 64.

Plus gênant encore, l'appareil n'a plus de lutins (images graphiques). Certes, le Vic-20 aussi en était dépourvu, mais ils jouaient un tel rôle dans les logiciels écrits pour le 64 qu'on s'était habitué à y voir une caractéristique essentielle de ce type d'ordinateur.

Dès sa mise sous tension, le 16 affiche le message familier aux usagers de la gamme Commodore, mais il est précisé que le BASIC employé est celui de la version 3,5 et qu'il y a 12 227 octets de mémoire disponible. En fait, cette version 3,5 est la cinquième du nom. La version originale, dite 1,0, ne comportait aucune commande permettant de gérer l'unité de disquettes; celle-ci n'était que très imparfaitement mise en œuvre par la version 2, que les responsables de Commodore ont eu la mauvaise idée d'implanter sur le Vic-20 et le 64.

Le Commodore 16
Le successeur du Vic-20 au sein de la gamme Commodore dispose d'une mémoire de 16 K et surtout d'un BASIC très amélioré. Il est compatible avec le Plus/4, mais pas avec le Vic-20 et le 64. (Cl. Chris Stevens.)

La version 4 fit son apparition en mars 1980 : très puissante et efficace, elle fut adaptée aux PETs à 80 caractères par ligne, le 8032, le 8096 et maintenant le 8296.

La version 3,5 est pratiquement identique, avec quelques commandes supplémentaires. En tout, le 16 possède environ cinquante instructions et fonctions de plus que le Vic-20. On citera, par exemple, certaines aides à la programmation lorsqu'on rédige ou débogue des programmes, des éléments de programmation structurée et des commandes graphiques et sonores.

MONITOR permet d'activer Tedmon, un moniteur langage machine intégré (qu'on peut aussi convoquer par SYS 4, comme sur les PETs). Il fait usage de mnémoniques réduits à une seule lettre : C compare deux parties de la mémoire et rend compte des différences; S effectue des sauvegardes sur cassette ou sur disquette.

Les routines du « kernal » (qui gèrent les entrées et les sorties) n'ont subi pratiquement aucune modification, à l'exception de la routine IOINIT propre au 64, qui permettait l'initialisation des périphériques (en particulier des cartouches), et qui a été implantée en $FF81.

Le graphisme

Le graphisme haute résolution fait usage d'une résolution de 320 × 160 pixels, qui tombe à 160 × 160 en mode multicolore. Il est infiniment plus facile de s'en servir que sur le 64, qui exigeait un nombre énorme de POKEs et de PEEKs. On peut même recourir à un système à deux fenêtres, bien que le texte n'y soit accepté qu'en bas d'écran, sur les cinq dernières lignes. Toutefois, en mode graphique, il est toujours possible de placer du texte où l'on veut, grâce à l'instruction CHAR; par exemple :

```
CHAR 1, 0, 0 «VOICI LA LIGNE DU HAUT»
```

imprime ce message tout en haut de l'écran. Le chiffre 1 correspond à la couleur choisie, et les deux zéros indiquent respectivement le numéro de la ligne et celui de la colonne. Il est possible d'afficher la phrase en vidéo inversée en ajoutant à l'instruction ,1 puis de revenir ensuite au mode normal avec ,0. Toute erreur de syntaxe, lorsque l'ordinateur est en mode GRAPHIC, renvoie l'usager en GRAPHIC 0, qui n'est autre que le mode texte proprement dit.

La commande DRAW est plus puissante que celle de l'Amstrad, mais moins sophistiquée que celle des ordinateurs MSX, qui fait un peu penser au LOGO. Il est très simple de tracer un carré grâce à :

```
DRAW, 10,10 TO 10,60 TO 60,60 TO 60,10 TO 10,10
```

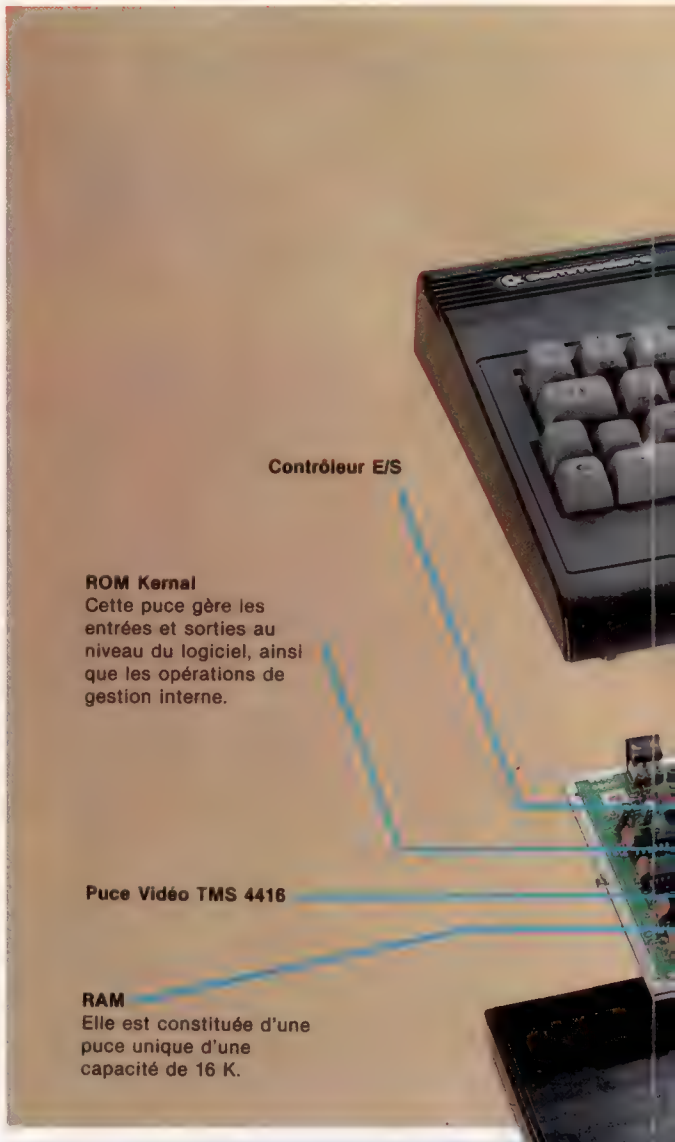
Dans ce cas particulier, la couleur n'a pas été précisée et ce sera donc celle de l'affichage à ce moment donné. Il est bien sûr toujours possible d'en changer. La commande BOX permet de tracer des rectangles (en spécifiant les coordonnées des quatre sommets) puis de les remplir de couleur.

La commande CIRCLE ne trace pas que des cercles, mais aussi des ellipses, des octogones, des formes en diamant ou en triangle, suivant les paramètres fournis. Les formes non circulaires sont tracées en spécifiant certains angles : 120° pour un triangle, 90° pour un diamant, 45° pour un octogone. La valeur implicite est de 2°. PAINT remplit les formes ainsi créées, la couleur étant celle du contour ou une autre choisie par l'usager. Ensuite, elles peuvent être sauvegardées sur disquette et rappelées grâce à SSHAPE et GSHAPE.

On dispose en tout de quinze couleurs (chacune ayant huit niveaux de luminosité), grâce auxquelles on peut colorier le fond, l'avant-plan ou la bordure d'écran; on compte aussi deux modes multicolores. Le 16 leur accorde une valeur par défaut de 7 (le niveau de luminosité le plus élevé). Dans tous les modes graphiques, les paramètres de couleur doivent être choisis dans l'une des cinq zones déjà définies.

Le son

Par rapport à la puce sonore SID *(Sound Interface Device)* du 64, les possibilités sonores du 16 (deux canaux seulement) semblent limitées, d'autant que nombre d'ordinateurs (Amstrad,



Ah, quand même!

Le BASIC du VIC-20 et du 64 a fait l'objet de telles critiques que Commodore s'est enfin décidé à doter ses nouvelles machines d'une version du langage comparable à celles dont disposent presque tous les micro-ordinateurs. Tous les enrichissements (graphisme, aide à la programmation, structuration des programmes, lecteur de disquette) sont les bienvenus. Il est regrettable que le 16 ne prenne pas en compte les procédures définies par l'utilisateur. Voici la liste des nouvelles commandes, incluant celles qui sont reprises du BASIC 4.

Fonctions	Aides progr.	Structure	Graphisme	Basic 4
RGR	AUTO	DO	GRAPHIC	DIRECTORY
RGCL	TRON	WHILE	SCNCLR	DSAVE
RLUM	TROF	LOOP	PAINT	DLOAD
JOY	HELP	UNTIL	CHAR	HEADER
RDOT	MONITOR	EXT	BOX	SCRATCH
INSTR	DELETE	ELSE	CIRCLE	COLLECT
DEC	RENUMBER	PUDEF	GSHAPE	COPY
HEX$	KEY	USING	SSHAPE	RENAME
SOUND	ERR$		DDRAW	BACKUP
VOL	TRAP		LOCATE	
	RESUME		COLOR	

Chris Stevens

COMMODORE 16

PRIX

★ ★

DIMENSIONS

76,2 × 203,2 × 406,4 mm.

UC

MOS 7501, 0,89 à 1,76 MHz.

MÉMOIRE

16 K de RAM (12 K accessible à l'utilisateur), 32 K de ROM (système d'exploitation et interpréteur BASIC compris).

ÉCRAN

Texte : 25 lignes de 40 caractères. Résolution graphique : 320 × 160 pixels. Cinq modes : texte, haute résolution, h.r. avec 5 lignes de texte, multicolore, multicolore avec 5 lignes de texte. 15 couleurs ayant 8 niveaux de luminosité.

INTERFACES

Port série Commodore, port d'extension et cartouche, prise cassette, deux ports manche à balai, sortie moniteur : composite/chrominance/luminosité/audio/RF avec poussoir réglable fin, prise d'alimentation (9 V).

LANGAGES DISPONIBLES

Interpréteur BASIC 3,5 implanté en ROM ; 75 commandes, notamment graphiques.

CLAVIER

66 touches de type machine à écrire, dont 7 touches de fonction programmables et une touche HELP.

POINTS FORTS

BASIC évolué, nombreuses commandes vers l'unité de disquettes, moniteur langage machine aisément accessible.

POINTS FAIBLES

Incompatibilité avec le reste de la gamme Commodore (sauf le Plus/4), donc très peu de logiciels disponibles.

MSX, etc.) sont aujourd'hui équipés d'une puce de General Instruments qui comporte trois canaux sonores (plus un de bruit blanc).

En revanche, le 16 n'a plus besoin des interminables séries de POKEs dont il fallait venir à bout sur le Commodore 64. Si vous connaissez la fréquence d'une note, il suffit de consulter le manuel pour y trouver le chiffre correspondant. Par exemple SOUND 1, 770, 60 donnera la note *la* (à une fréquence de 440 Hz), d'une durée de 1 s sur le canal 1.

L'ambitus est de quatre octaves (de 110 à 1 575 Hz). On peut se servir simultanément des deux canaux musicaux (1 et 2), ou de l'un d'eux seulement, et du canal bruit blanc (3). Le signal de sortie combine les deux sons produits ; il sera donc uniquement mono, malheureusement.

Le 16 est un appareil très intéressant, au BASIC évolué et aux puissantes commandes graphiques, mais ses capacités sonores sont vraiment réduites, même par rapport au Vic-20. Il est vrai qu'elles sont d'un emploi aisé. Les logiciels qui lui sont destinés restent encore très rares, et c'est une situation qui est amenée à se prolonger tant que le 16 n'aura pas percé sur le marché. Il est de surcroît très regrettable qu'aucun programme prévu pour le Vic-20 ou le 64 ne puisse tourner sur cette machine.

Le fil de l'histoire

Les deux jeux d'aventures que nous créons sont de type « textuel » : le programme affiche à l'écran la description du lieu. Mettons au point un utilitaire pour formater ces messages.

Digitaya et la Forêt hantée sont des aventures « textuelles » : elles recourent aux mots pour décrire les lieux et les événements. On se sert pour cela d'instructions PRINT, mais si le procédé est commode, il lui arrive de manquer d'élégance : dès que l'une de ces instructions dépasse la longueur d'une ligne d'écran, elle est coupée en deux; la fin du message est renvoyée à la ligne suivante, et un mot peut se retrouver séparé en deux tronçons. Il est toujours possible, bien que très fastidieux, de tout formater « à la main », mais la besogne, peu exaltante, devient franchement exaspérante dans un jeu d'aventures, où les messages sont nombreux. Une solution bien meilleure consiste à rédiger une routine de formatage de texte. Les phrases lui seront transmises par l'intermédiaire de chaînes de caractères, et elle assurera automatiquement leur affichage à l'écran.

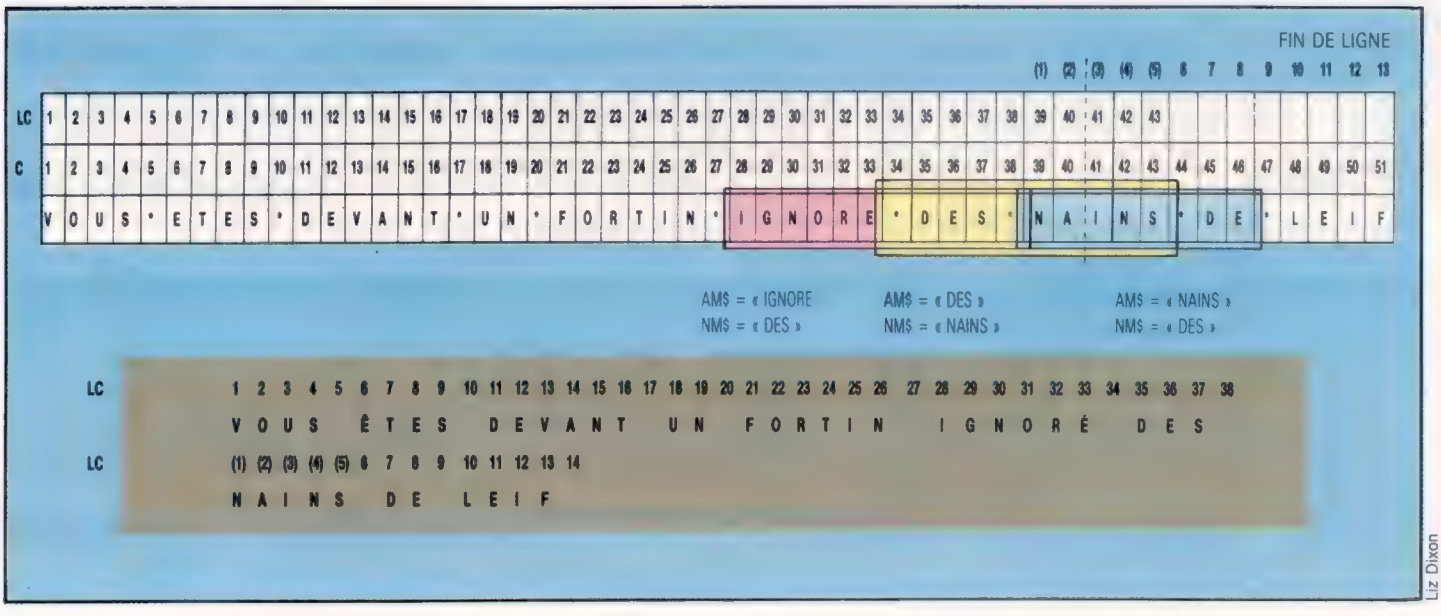
Digitaya et la Forêt hantée font tous deux usages d'une routine spécifique. Aussi verrons-nous d'abord comment elle fonctionne avant d'en revenir aux problèmes de programmation proprement dits. Voici le listage de la Forêt hantée :

```
5500 REM ***** S/R FORMATAGE ECRAN *****
5510 LC=0 : REM COMPTE CARACTERES DE LA LIGNE
5520 DM=1 : REM VALEUR INITIALE DEBUT MOT
5530 AM$="" : REM VALEUR INITIALE MOT DE DEPART
5540 LL=40 : REM LONGUEUR DE LA LIGNE
5550 PH$=PH$+" BIDON "
5560 PRINT
5570 FOR C=1 TO LEN(PH$)
5580 LC=LC+1
5590 IF MID$(PH$,C,1)=" " THEN GOSUB 5800
5600 NEXT C
5605 PRINT
5610 RETURN
5620 :
5800 REM ***** S/R VERIFICATION FIN DE LIGNE *****
5810 NM$=MID$(PH$,DM,C-DM+1): REM NOUVEAU MOT
5820 IF LC<LL THEN PRINT AM$; : GOTO 5840
5830 PRINT AM$ : LC=LEN(NM$)
5840 DM=DM+1 : AM$=NM$
5850 RETURN
```

Formation d'images
Le programme pour formater l'écran employé dans la Forêt hantée et Digitaya autorise toute sortie écran. En utilisant les variables AM$ et NM$, le programme « voit » le premier mot de la phrase à écrire. Si le mot suivant est susceptible de faire dépasser la longueur autorisée, les deux points supprimant le retour-chariot est omis, provoquant une nouvelle ligne.

Notre routine recherche d'abord tous les blancs dans la phrase qui lui a été transmise par la variable PH$. A chaque découverte d'un espace, le contrôle passe à la sous-routine de la ligne 5800, qui gère plusieurs tâches importantes. En employant DM (dont la valeur initiale est de 1) pour indiquer le début d'un mot, et C pour savoir quel est le caractère en cours d'examen, elle isole le mot rencontré avant l'espace grâce à MID$, puis le place en mémoire à l'aide de NM$ (Nouveau Mot). Avant que le contenu de NM$ soit affiché à l'écran, il est transféré à AM$.

LC compte combien de caractères ont été utilisés jusque-là dans la ligne; l'instruction ligne 5820 s'assure que sa valeur est toujours inférieure à celle de LL (longueur maximum de la ligne de texte). Si c'est le cas, AM$ est affiché, suivi d'un point-virgule pour que tout ce qui sera imprimé ensuite vienne sur la même ligne. Si LC est supérieur à LL, AM$ sera bel et bien affiché, mais, cette fois, sans point-virgule, ce qui veut dire que tout ce qui viendra ensuite apparaîtra sur la ligne



suivante. De plus, LC prend une valeur qui est celle du nouveau mot.

Voyons maintenant comment cette sous-routine fonctionne dans la pratique. Elle recherche un ou des espace(s) dans la phrase qu'elle analyse. Quand elle en a trouvé un, tous les caractères compris entre cet espace et celui qui le précède sont considérés comme formant un nouveau mot. En fait, la routine a un mot d'avance par rapport à l'affichage. Avant d'en ajouter un autre, elle vérifie si le nombre maximum de caractères par ligne a déjà été atteint. Si tel est le cas, elle commence une nouvelle ligne, ce qui évite la fragmentation de tel ou tel mot. « BIDON » a aussi son importance : il constitue le dernier mot placé en NM$, et les espaces qui l'entourent sont décisifs ; le premier permet de l'identifier comme mot isolé, et le second représente le dernier espace que repérera la routine.

Prenons comme exemple la phrase : « Vous êtes devant un fortin ignoré des nains de Leif ». Chaque ligne d'écran ne peut accepter que quarante caractères au maximum. Sans formatage, le mot « nains » serait coupé en deux, et « ins » se retrouverait au début de la ligne suivante. Toutefois, notre routine analyse la phrase à raison de deux mots à la fois. Si nous considérons les deux mots qui précèdent « nains », nous voyons que « ignoré » est stocké en AM$ et « des » en NM$. Après avoir vérifié que LC n'excède pas 40, le programme affiche à l'écran AM$ (instruction PRINT suivie d'un point-virgule). « Des » passe ensuite de NM$ à AM$ tandis que la routine identifie « nains » ; LC est désormais supérieur à 40, et AM$ est donc affiché à son tour, mais cette fois l'instruction est dépourvue de point-virgule. « Nains » est donc imprimé sur la ligne suivante. LC, de son côté, prend pour valeur le nombre de caractères contenus dans ce mot.

Mise à l'épreuve

Afin de tester notre routine, nous nous en servirons pour formater et afficher le tout début de l'histoire. Nous pouvons ainsi mettre en place une phrase comprenant jusqu'à 248 caractères, grâce à la variable PH$, puis procéder automatiquement au formatage. Tapez donc les lignes suivantes :

```
1000 REM ***** C'EST PARTI ! *****
1010 PH$= "BIENVENUE DANS LA FORET HANTEE'"
1020 GOSUB 5500 : REM FORMATAGE
1030 PRINT
1040 PH$="VOUS SORTEZ D'UN PROFOND SOMMEIL"
1050 PH$= PH$+", LE SOL EST DOUX ET SEC. "
1060 PH$= PH$+"VOUS NE SAVEZ COMMENT VOUS ETES
     ARRIVE LA.
1070 PH$= PH$+"MAIS VOUS SAVEZ QU'IL VOUS FAUT"
1080 PH$= PH$+"TROUVER LE VILLAGE PRES DU BOIS POUR"
1090 PH$= PH$+"ETRE EN SURETE."
1100 GOSUB 5500 : REM FORMATAGE
1110 PRINT
1120 PH$= "VOUS REGARDEZ AUTOUR DE VOUS,
     CHERCHANT DES YEUX VOS MAIGRES BIENS."
1130 GOSUB 5500 : REM FORMATAGE
1140 PRINT : PRINT "APPUYER SUR UNE TOUCHE
     POUR COMMENCER."
1150 GET A$ : IF A$="" THEN 1150
1160 PRINT CHR$(147) : REM EFFACE L'ECRAN
1170 RETURN
```

```
205 GOSUB 1000: REM STORY SO FAR
990 END
```

Listage Digitaya

```
1110 GOSUB 1250 : REM C'EST PARTI '
1270 END

1290 REM ***** C'EST PARTI ! *****
1300 PH$= "BIENVENUE A 'DIGITAYA'"
1310 GOSUB 5800 : REM FORMATAGE
1320 PRINT
1330 PH$= "TANDIS QUE LA MACHINE BOURDONNE, VOUS
     REGARDEZ AUTOUR"
1340 PH$= PH$+" DE VOUS. AU SUD ET AU NORD
     S'ETEND UNE GRANDE AUTOROUTE."
1350 PH$= PH$+"VOUS DEVEZ RETROUVER 'DIGITAYA'
     ET L'ACCOMPAGNER"
1360 PH$= PH$+" JUSQU'A UN DES PORTS DE SORTIE."
1370 PH$= PH$+"...MAIS LEQUEL ?"
1380 GOSUB 5800
1390 PRINT : PRINT "APPUYEZ SUR UNE TOUCHE POUR
     COMMENCER "
1400 GET A$ : IF A$=""THEN 1400
1410 PRINT CHR$(147) :REM EFFACE L'ECRAN
1420 RETURN


5800 REM ***** S/R FORMATAGE ECRAN *****
5890 LC=0 : REM COMPTE CARACTERES DE LA LIGNE
5900 DM=1 : REM VALEUR INITIALE DEBUT MOT
5910 AM$="" :REM VALEUR INITIALE MOT DE DEPART
5920 LL=40 :REM LONGUEUR DE LA LIGNE
5930 PH$= PH$+" BIDON "
5940 PRINT
5950 FOR C=1 TO LEN(PH$)
5960 LC=LC+1
5970 IF MID$(PH$,C,1)= " " THEN GOSUB 6020
5980 NEXT C
5990 PRINT
6000 RETURN
6010 :
6020 REM ***** S/R VERIF. FIN DE LIGNE *****
6030 NM$=MID$(PH$,DM,C-DM+1)
6040 IF LC  LL THEN PRINT AM$;:GOTO 6060
6050 PRINT AM$:LC=LEN(NM$)
6060 DM=DM+1:AM$=NM$
6070 RETURN
```

Variantes de basic

Spectrum :

Pour Digitaya, procédez aux changements suivants dans la routine de formatage :

Remplacez PH$ par P$, AM$ par A$, NM$ par N$

```
5920 LET LL=32 : REM LONGUEUR DE LA LIGNE
5970 IF P$ (C TO C)=« » THEN GOSUB 6020
6030 LET N$=P$(DM TO C)
```

Dans la routine C'est Parti!, remplacez PH$ par P$

```
1400 IF INKEY$=« » THEN 1400
1410 CLS
```

Pour la Forêt hantée, remplacez les mêmes noms de variables, et modifiez les lignes suivantes :

```
5540 LET LL=32 : REM LONGUEUR DE LIGNE
5590 IF P$(C TO C)=« » THEN GOSUB 5800
5810 LET N$=P$(DM TO C)
```

ainsi que

```
1150 IF INKEY$=« » THEN 1150
1160 CLS
```

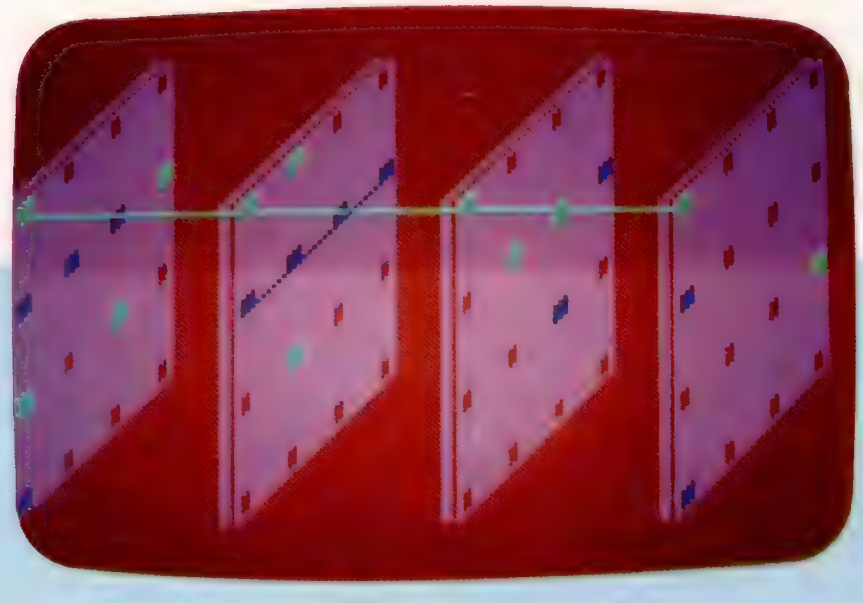
Morpion en trois axes

Ce jeu de Roland Mariott nécessite 24 K de mémoire. Il vous permet de jouer contre l'ordinateur Atari ou contre un adversaire de votre choix.

Pour gagner, il faut réussir à placer quatre carrés de même couleur dans une rangée. Vous jouez dans quatre plans (A,B,C,D) divisés eux-mêmes en seize carrés. Pour jouer, choisissez d'abord le plan puis tapez un nombre correspondant au carré choisi. Ils sont numérotés comme suit :

13 14 15 16
09 10 11 12
05 06 07 08
01 02 03 04

Voici des exemples de coups (n'oubliez pas qu'il est inutile de taper la touche « Return ») : D15, A05, B10, B16, C02. Lorsque l'ordinateur vous demande le nombre de joueurs, tapez 1 pour jouer contre lui ou 2 pour affronter un autre adversaire.



```
0 REM ********************************
1 REM * MORPION EN TROIS DIMENSIONS *
2 REM *      DE ROLAND MARRIOTT      *
3 REM ********************************
4 GOSUB 3000
5 DIM SPACE$(64,2):DIM MARK(64):DIM BLOB$
86,7*:DIM HZ(40)
6 CL=0
7 GOSUB 6000
10 GRAPHICS 7+16:COLOR 1
17 SETCOLOR 2,12,8:SETCOLOR 4,0,3
18 SETCOLOR 1,4,8:SETCOLOR 0,2,6:SETCOLO
R 3,3,8
20 FOR X=0 TO 129 STEP 43
30 FOR Y=95 TO 30 STEP -1
40 PLOT X,Y:DRAWTO X+30,Y-30
50 NEXT Y
60 NEXT X
61 PLOT 0,28:DRAWTO 28,0:PLOT 0,27:DRAWT
O 27,0
62 FOR RT=0 TO 86 STEP 43
63 PLOT RT+41,95:DRAWTO RT+41,30:DRAWTO
RT+71,0:PLOT RT+40,95:DRAWTO RT+40,30:DR
AWTO RT+70,0
64 NEXT RT
65 COLOR 0:SETCOLOR 4,9,2
66 FOR Z=0 TO 129 STEP 43
70 FOR Y=91 TO 31 STEP -20
72 FOR X=1 TO 31 STEP 9
74 FOR A=0 TO 1
760 PLOT Z+X+A,Y-A-X+3:DRAWTO Z+X+A,Y-A-
X
780 NEXT A
850 NEXT X
900 NEXT Y
950 NEXT Z
1000 FOR P=1 TO 64:READ H,J
1010 SPACE$(P,1)=H:SPACE$(P,2)=J
1012 NEXT P
1015 IF PEEK(764)=255 THEN 1015
1016 CLOSE #1:OPEN #1,4,0,"K:":GET #1,W
1017 IF W<65 OR W>88 THEN POKE 764,255:G
OTO 1015
1018 IF W=65 THEN W=0:IF W=66 THEN W=16:
IF W=67 THEN W=32
1019 IF W=66 THEN W=16
1020 IF W=67 THEN W=32
1021 IF W=68 THEN W=48
1023 POKE 764,255
1024 IF PEEK(764)=255 THEN 1024
1025 CLOSE #5:OPEN #5,4,0,"K:":GET #5,Z
1026 IF Z<48 OR Z>49 THEN POKE 764,255:G
OTO 1024
1027 POKE 764,255
1028 CLOSE #4:OPEN #4,4,0,"K:":GET #4,WW
:IF WW<48 OR WW>57 THEN POKE 764,255:GOT
O 1027
1033 REM FOR TR=0 TO 2
1034 SP=WW-16*(Z-48)+WW-48:LOCATE SPACE$(S
P,1),SPACE$(SP,2),GW:IF GW<>0 THEN GOTO 1
015
1035 CL=CL+1:IF CL=2 THEN CL=0
1036 FOR RY=15 TO 0 STEP -1:COLOR (RY+CL
+2):SETCOLOR 1,3,4
1037 FOR TR=0 TO 2
1038 PLOT SPACE(SP,1)+TR,SPACE(SP,2)-TR:
DRAWTO SPACE(SP,1)+TR,SPACE(SP,2)-TR-3
1039 NEXT TR:NEXT RY
1040 CLOSE #1
1042 FOR PQ=1 TO 64
1044 LOCATE SPACE(PQ,1),SPACE(PQ,2),V
1046 MARK(PQ)=V:NEXT PQ
1050 C=CL+2
1100 A=1:B=16:D=16:E=1:GOSUB 1400
1105 A=1:B=61:D=1:E=4:GOSUB 1400
1110 A=1:B=13:D=17:E=4:GOSUB 1400
1115 A=4:B=16:D=15:E=4:GOSUB 1400
1120 A=1:B=4:D=20:E=1:GOSUB 1400
1125 A=13:B=16:D=12:E=1:GOSUB 1400
1130 A=1:B=1:D=21:E=1:GOSUB 1400
1135 A=13:B=13:D=13:E=1:GOSUB 1400
1140 A=16:B=16:D=11:E=1:GOSUB 1400
1145 A=4:B=4:D=19:E=1:GOSUB 1400
1150 A=1:B=49:D=5:E=16:GOSUB 1400
1155 A=4:B=52:D=3:E=16:GOSUB 1400
1160 A=1:B=4:D=4:E=1:GOSUB 1400
1165 A=17:B=20:D=4:E=1:GOSUB 1400
1170 A=33:B=36:D=4:E=1:GOSUB 1400
1175 A=33:B=36:D=4:E=1:GOSUB 1400
1180 A=49:B=52:D=4:E=1:GOSUB 1400
1299 IF JOUEUR=1 AND CL<>0 THEN GOTO 150
0:REM IF CL=0 THEN GOTO 1015
1300 GOTO 1015
1400 FOR PQ=A TO B STEP E
1410 IF MARK(PQ)=C AND MARK(PQ+D)=C AND
MARK(PQ+2*D)=C AND MARK(PQ+3*D)=C THEN G
OSUB 1450
1420 NEXT PQ
1430 RETURN
1450 PLOT SPACE(PQ,1)+1,SPACE(PQ,2)-1:DR
AWTO SPACE(PQ+3*D,1)+1,SPACE(PQ+3*D,2)-1
:RETURN
1500 REM JEU ORDINATEUR
1600 FOR PQ=1 TO 64
1605 LOCATE SPACE(PQ,1),SPACE(PQ,2),V
1606 IF V=2 THEN V=10
1608 IF V=3 THEN V=50
1610 MARK(PQ)=V:NEXT PQ
1615 H=0
1620 A=1:B=16:D=16:E=1:GOSUB 1700
1625 A=1:B=61:D=1:E=4:GOSUB 1700
1630 A=1:B=13:D=17:E=4:GOSUB 1700
1635 A=4:B=16:D=15:E=4:GOSUB 1700
1640 A=1:B=4:D=20:E=1:GOSUB 1700
1645 A=13:B=16:D=12:E=1:GOSUB 1700
1650 A=1:B=1:D=21:E=1:GOSUB 1700
1655 A=13:B=13:D=13:E=1:GOSUB 1700
1660 A=16:B=16:D=11:E=1:GOSUB 1700
1665 A=4:B=4:D=19:E=1:GOSUB 1700
1670 A=1:B=49:D=5:E=16:GOSUB 1700
1675 A=4:B=52:D=3:E=16:GOSUB 1700
1680 A=1:B=4:D=4:E=1:GOSUB 1700
1682 A=17:B=20:D=4:E=1:GOSUB 1700
1686 A=33:B=36:D=4:E=1:GOSUB 1700
1688 A=49:B=52:D=4:E=1:GOSUB 1700
1690 GOTO 1740
1700 FOR PQ=A TO B STEP E
1705 H=H+1
1710 SUM=MARK(PQ)+MARK(PQ+D)+MARK(PQ+2*D
)+MARK(PQ+3*D)
1720 IF SUM=30 THEN GOTO 1830
1725 BLOB(H,1)=SUM:BLOB(H,2)=PQ:BLOB(H,3
)=D
1730 NEXT PQ:RETURN
1740 FOR TZ=1 TO H:IF BLOB(TZ,1)=150 THE
N GOTO 1850
1745 NEXT TZ
1746 YZ=0:FOR TZ=1 TO H
1748 IF BLOB(TZ,1)=20 OR BLOB(TZ,1)=100
THEN GOSUB 1860
1750 NEXT TZ:IF YZ<>0 THEN GOTO 1760
1755 GOTO 2050
1760 FOR TZ=1 TO YZ:FOR PZ=2 TO (YZ-1)
1765 IF HZ(TZ)=HZ(PZ) AND TZ<>PZ THEN SP
=HZ(TZ):GOTO 1035
1766 NEXT PZ:NEXT TZ
1770 IF YZ<>0 THEN SP=HZ(YZ):GOTO 1035
1825 GOTO 2050
1830 FOR TK=0 TO 3:IF MARK(PQ+TK*D)=0 TH
EN SP=PQ+TK*D:GOTO 1035
1835 NEXT TK
1850 REM DEFENSE
1855 FOR TK=0 TO 3:IF MARK(BLOB(TZ,2)+TK
*BLOB(TZ,3))=0 THEN SP=BLOB(TZ,2)+TK*BLO
B(TZ,3):GOTO 1035
1857 NEXT TK
1860 FOR P=0 TO 3
1863 IF MARK(BLOB(TZ,2)+P*BLOB(TZ,3))=0
THEN YZ=YZ+1:HZ(YZ)=BLOB(TZ,2)+P*BLOB(TZ
,3)
1865 NEXT P:RETURN
1890 GOTO 2050
1950 GOTO 2050
2050 SP=INT(64*RND(1)+1):IF MARK(SP)=0 T
HEN GOTO 1035
2055 GOTO 2050
3000 REM ECRAN ENTIER
3020 GRAPHICS 2+16:SETCOLOR 4,13,2
3030 POSITION 3,1:? #6;"################"
3040 POSITION 3,2:? #6;"#  MORPION EN #"
3050 POSITION 3,3:? #6;"# 3 DIMENSIONS#"
3052 POSITION 3,4:? #6;"#  DE ROLAND  #"
3054 POSITION 3,5:? #6;"#   MARRIOTT   #"
3060 POSITION 3,6:? #6;"################"
3070 POSITION 1,7:? #6;"    TAPEZ START"
3080 IF PEEK(53279)=6 THEN RETURN
3090 GOTO 3080
5000 DATA 1,93,10,84,19,75,28,66
5010 DATA 1,73,10,64,19,55,28,46
5020 DATA 1,53,10,44,19,35,28,26
5030 DATA 1,33,10,24,19,15,28,6
5040 DATA 44,93,53,84,62,75,71,66
5050 DATA 44,73,53,64,62,55,71,46
5060 DATA 44,53,53,44,62,35,71,26
5070 DATA 44,33,53,24,62,15,71,6
5080 DATA 87,93,96,84,105,75,114,66
5090 DATA 87,73,96,64,105,55,114,46
5100 DATA 87,53,96,44,105,35,114,26
5110 DATA 87,33,96,24,105,15,114,6
5120 DATA 130,93,139,84,148,75,157,66
5130 DATA 130,73,139,64,148,55,157,46
5140 DATA 130,53,139,44,148,35,157,26
5150 DATA 130,33,139,24,148,15,157,6
6000 GRAPHICS 2+16:? #6;"  2 joueurs OU"
6010 ? #6;"1 JOUEUR CONTRE MOI"
6020 ? #6;" (TAPEZ 1 OU 2 )"
6030 OPEN #1,4,0,"K:":GET #1,M:IF M<49 O
R M>50 THEN POKE 764,255:CLOSE #1:GOTO 6
030
6040 JOUEUR=M-48:CLOSE #1
6050 RETURN
```

Points d'interruption

Dans cette suite de notre programme de « débogage », nous allons compléter le module de routines pour manier les points d'interruption.

Nous allons définir deux sous-programmes pour le module d'interruption — l'un pour enlever des interruptions, et l'autre pour restaurer l'opc initial où nous avons placé un opc temporaire SWI. La première routine qu'il nous faut considérer est appelée Suppression-d'interruption (de la table d'interruptions).

Jusqu'à 16 points d'interruption ont été admis dans la table d'interruptions (BPTAB). Pour en enlever un, il nous faut son numéro comme décalé (compris entre 0 et 15) dans cette table. L'entrée de table est supprimée en décalant toutes les entrées suivantes de la table d'une case (deux octets) et en décrémentant le nombre d'interruptions.

Suppression d'interruption

Données :
Nombre-d'interruptions est une valeur 8 bits.
Numéro-d'interruption est un compteur 8 bits.
Table-d'interruption est un tableau d'adresses 16 bits.
Entrée-à-supprimer est un décalé 8 bits (de valeur comprise entre 1 et 16).
Traitement : Supprimer-interruption
Décrémenter Nombre-d'interruptions.
Si entrée-à-supprimer < = nombre-d'interruptions (avant-dernier) ALORS.
Pour Numéro-d'interruption = Entrée-à-supprimer jusqu'à Nombre-d'interruptions (avant-dernier).
Déplacer Table-d'interruption (Numéro-d'interruption + 1) à Table-d'interruption (Numéro d'interruption).
Déplacer Valeurs-supprimées (Numéro-d'interruption + 1) à Valeurs-supprimées (Numéro-d'interruption).
FinPour
FinSi
Fin du traitement

Le paramètre entrée-à-supprimer peut être passé en B. Le compteur numéro-d'interruption peut alors aussi être placé en B, et donnera automatiquement sa valeur initiale correcte. Après l'avoir comparé avec le nombre-d'interruptions, il doit être décrémenté pour former le décalé dans la table de valeurs-supprimées à 8 bits, puis être décalé (multiplié par 2) pour former un décalé dans la table-d'interruptions à 16 bits.

Nous pouvons conserver le décalé à 8 bits dans B et le décalé à 16 bits dans A. Les adresses des entrées dans les deux tables peuvent être dans X et Y, de sorte que nous pouvons utiliser l'auto-incrémentation pour parcourir la table. L'entrée à 16 bits peut être décalée par U, mais celle à 8 bits devra encore utiliser A.

Le dernier traitement employé dans ce module déplace physiquement une interruption en remplaçant l'opc SWI par l'opc initial de la table des valeurs-supprimées.

Restauration d'interruption

Données
Numéro-d'interruption est le décalé à 8 bits dans la Table-d'interruption.
Traitement
Prendre valeur dans valeurs-supprimées (numéro-d'interruption).
La stocker dans adresse dans table-d'interruption (numéro-d'interruption).

Nous supposerons que le paramètre numéro-d'interruption est passé en B sous la forme usuelle comme un nombre compris entre 1 et 16 qui doit être converti pour fonctionner comme décalé dans les tables.

Nous en sommes arrivés au stade de la construction d'un module pour exécuter les huit commandes à une lettre qui gèrent le système. Certaines de ces commandes peuvent être exécutées directement par les routines que nous avons déjà écrites. Cependant, dans l'intérêt de la structure, nous incorporerons des appels à celles-ci à partir du module.

La commande B, pour insérer une interruption, est complètement couverte par la routine insertion-d'interruption (BP01). Dans ce module, il nous faudra donc simplement :

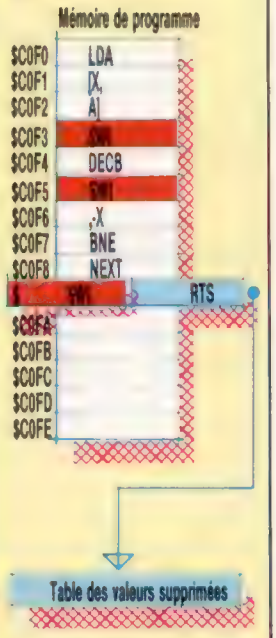
```
CMDB BRA BP01
```

La commande U, pour supprimer une interruption, est presque couverte par la routine que nous venons d'écrire (BP04). Cependant, il nous faudra d'abord l'adresse de l'interruption à supprimer, laquelle se trouve dans la table-d'interruption. Sinon, nous ignorons cette commande; si elle y est, alors nous pouvons passer le décalé au sous-programme en BP02.

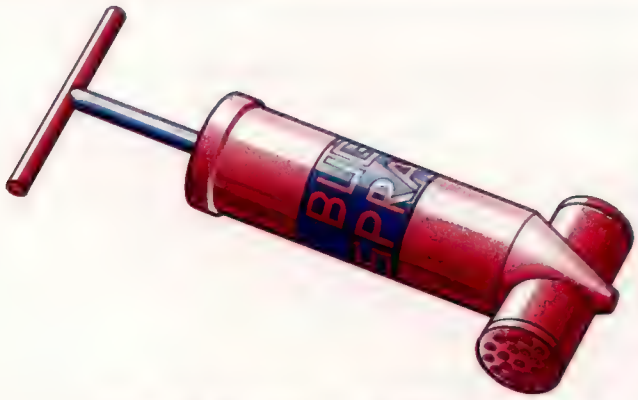
Commande U

Données
Suggestion à afficher.
Adresse-d'interruption est l'entrée.
Table-d'interruption.
Numéro-d'interruption.
Traitement
Affiche suggestion.
Entre adresse-d'interruption.
Met numéro-d'interruption à 16.

Interrompre le code



Le débogueur insère des interruptions dans le code objet en question, en commençant par sauvegarder le code à partir de cette adresse dans la table des valeurs-supprimées et en incrémentant le compteur nombre-d'interruptions, puis en écrivant dans l'octet de l'interruption l'opc SWI. La table des valeurs-supprimées ressemble à une pile, mais c'est en fait un amoncellement tel, que l'on peut en sortir les valeurs de n'importe quel octet, et pas seulement le dernier octet entré. Lorsqu'une interruption est supprimée, la valeur-supprimée correspondante est copiée de la table dans la mémoire de programme, et l'octet redondant est éliminé en décalant tous les octets de la table au-dessus de lui en mémoire, et enfin en décrémentant le compteur nombre-d'interruptions.



```
Tant que table-d'interruption (numéro-d'interruption) <> Adresse-
d'interruption et numéro-d'interruption > 0.
Décrémenter numéro-d'interruption.
Si trouvé alors.
Supprimer-interruption.
```

L'adresse-d'interruption peut être conservée en Y, laissant X disponible pour un pointeur dans la table. Le numéro-d'interruption peut être gardé en B.

La commande D, pour afficher les interruptions, est couverte par la routine DISPBP (Display-Breakpoints). On y accède simplement par une branche de sous-programme :

```
CMDD BRA DISPBP
```

La commande S, pour commencer l'exécution du programme, est plus compliquée, puisque c'est là que doivent être insérées les interruptions. L'opc pour l'instruction SWI doit être insérée à chaque adresse de la table-d'interruption, et l'opc qui y est déjà est mis dans valeurs-supprimées. Lorsque c'est fait, le contrôle doit être transféré à l'adresse de départ du programme. Il faut aussi noter que l'interruption suivante a le numéro 1. Voici tout le traitement pour le début du programme :

Commande S

Données

Nombre-d'interruptions est une valeur à 8 bits.
Table-d'interruption.
Valeurs-supprimées.
Numéro-d'interruption est un compteur à 8 bits.
Interruption-suivante est une valeur à 8 bits.
SWI est un opc à 8 bits.
Adresse-de-départ est l'adresse à 16 bits où commence le programme que nous débogons.

Traitement

```
Met Numéro-d'interruption à Nombre-d'interruptions.
Tant que Numéro-d'interruption >0.
  Mettre-interruption (Numéro-d'interruption).
  Décrémente Numéro-d'interruption.
FinTant.
Met Interruption-suivante à 1.
Saut à Adresse-de-départ.
```

Pour ce faire, nous utilisons la routine Mettre-interruption — déjà codée — qui requiert le Numéro-d'interruption (moins un, de sorte qu'il peut être utilisé comme décalé dans les tables) dans A. Il sera plus commode de décrémenter A avant l'appel à Mettre-interruption. L'ensemble de la routine codée est donné ici.

La fin de la routine nécessite une petite explication. Lorsque le programme à tester est exécuté, nous n'avons besoin d'aucun autre article sur la pile, et nous devons donc nous assurer que la pile est vide lorsque le contrôle est transféré au programme. Nous pouvons effacer les articles superflus de la pile dans le module principal, mais si cette routine est appelée par BSR (pour la cohérence avec les autres commandes), l'adresse de retour devra se trouver sur la pile. Si nous l'y laissons, alors, dans une longue séance (où le programme devra être recommencé à plusieurs reprises) la pile continuera à croître. Notre solution supprime les adresses de la pile en même temps qu'elle retransfère le contrôle au programme, et cela, en remplaçant l'adresse de retour sur la pile par l'adresse de départ. RTS fait alors sortir l'adresse de retour, qui est maintenant l'adresse de départ de la pile et transfère ainsi le contrôle tout en réinitialisant la pile.

La commande finale que nous verrons ici est M : elle sert à inspecter et à changer des emplacements mémoire. L'idée est d'entrer une adresse et d'afficher son contenu à l'écran. L'utilisateur peut ensuite entrer un autre nombre à deux chiffres hex à placer à cette adresse, ou simplement un retour. Dans les deux cas, nous allons à l'emplacement mémoire suivant. L'utilisateur peut arrêter ce processus en entrant un point. La routine GETHX2 a été faite pour cela, en permettant d'entrer deux chiffres hex, un point ou un retour.

Commande M

Données

Emplacement-en-cours est l'adresse 16 bits de l'emplacement inspecté.
Valeur-en-cours se trouve dans Emplacement-en-cours; 8 bits.
Nouvelle-valeur pour Emplacement-en-cours; 8 bits.

Traitement

```
Entre emplacement-en-cours.
Répète.
  Affiche valeur-en-cours.
  Entre nouvelle-valeur.
  Si nouvelle-valeur n'est pas un point alors.
    Si nouvelle-valeur n'est pas un retour alors.
      Stocke nouvelle-valeur dans emplacement-en-cours.
    FinSi
    Incrémente emplacement-en-cours.
    Affiche emplacement-en-cours.
    FinSi
Jusqu'à ce que emplacement-en-cours soit un point.
```

Pour cette routine, emplacement-en-cours est stocké en X, et le registre B sert à la fois pour valeur-en-cours et nouvelle-valeur. A sert de drapeau pour indiquer laquelle des trois possibilités (nombre hex, point ou retour) a été entrée.

Il nous faut maintenant concevoir et coder les trois dernières commandes — G, R et Q. Celles-ci impliquent l'emploi d'un mécanisme d'interruption que nous avons déjà vu. Nous y reviendrons prochainement ainsi que sur la conception du module principal de débogage.

Routine de suppression

```
BP04    PSHS    A,B,X,Y,U       Sauvegarde les registres utilisés
        DEC     NUMBP,PCR       Décrémente nombre-d'interruptions

IF02    CMPB    NUMBP,PCR       Si entrée-à-supprimer <
        BGT     ENDF02          Nombre-d'interruptions
        DECB                    Convertit B en décalé
        TFR     B,A             Copie B dans A
        LSLA                    Convertit A en décalé
        LDX     BPTAB,PCR       Adresse de base de table-d'interruptions
        LEAX    A,X             Table-d'interruption (numéro-d'interruption)
        LDY     REMTAB,PCR      Adresse de base de valeurs-supprimées
        LEAY    B,Y             Valeurs-supprimées (numéro-d'interruption)
FOR00   LDU     2,X             Entre table-d'interruption à décaler
        STU     ,X++            Recule d'une position
        LDA     1,Y             Entre valeur-supprimée
        STA     ,Y+             Recule d'une position
        INCB
        CMPB    NUMBP,PCR       Dernier?
        BLT     FOR00           Suivant
ENDF02  PULS    A,B,X,Y,U,PC    Restaure et Retour
```

Routine de restauration d'interruption

```
BP05    PSHS    A,B,X
        DECB                    Convertit en décalé dans valeurs-supprimées
        LDX     REMTAB,PCR      Adresse de base de valeurs-supprimées
        LDA     B,X             Entre valeur à supprimer
        LSLB                    Convertit en décalé dans table-d'interruption
        LDX     BPTAB,PCR       Adresse de base de table-d'interruption
        STA     [B,X]           Stocke à l'adresse dans table
        PULS    A,B,X,PC        Restaure et Retour
```

Commande U

```
PROMPT  FCB     '>
CMDU    PSHS    A,B,X,Y         Sauvegarde registres utilisés
        LDA     PROMPT,PCR
        BSR     OUTCH           Affiche suggestion
        BSR     GETADD          Entre adresse
        TFR     D,Y             Met adresse-d'interruption en Y
        LDB     MAXBP,PCR       Nombre maximum d'interruptions (16)
        LDX     BPTAB,PCR       Adresse de base de table-d'interruption
        TFR     B,A
        LSLA                    A est le décalé à la fin de la table-d'interruption
        LEAX    A,X             X pointe après la fin de la table
        TSTB                    Met drapeaux sur contenu de B
WHIL02  BLE     ENDW02          Tant que B>0
        CMPY    ,--X            (Rappelez-vous que X est d'abord décrémenté)
        BEQ     ENDW02          et Adresse-d'interruption n'est pas trouvé dans la table
        DECB                    Décrémente Numéro-d'interruption
        BRA     WHIL02
ENDW02  TSTB                    Trouvé si B>0
IF03    BLE     ENDF            Si trouvé alors
        BSR     BP04            Supprimer-interruption
ENDF03  PULS    A,B,X,Y
```

Commande S

```
START   RMB     2               Adresse-de-départ
CMDS    LDA     NUMBP,PCR       Met numéro-d'interruption à nombre-d'interruptions
WHIL03  TSTA                    Teste la valeur de numéro-d'interruption
        BLE     ENDW03          Tant que numéro-d'interruption>0
        DECA                    Décrémente numéro-d'interruption
        BSR     BP02            Met-interruption
        BRA     WHIL03          Interruption-suivante
ENDW03  LDA     #1              Met interruption-suivante à 1
        STA     NEXTBP,PCR
        STD     1,S
        RTS
```

Commande M

```
PROMPT  FCB     '>
SPACE   FCB     32              Code ASCII pour espace
CMDM    PSHS    A,B,X           Sauvegarde registres utilisés
        LDA     PROMPT,PCR
        BSR     OUTCH           Affiche suggestion
        BSR     GETADD          Entre emplacement-en-cours
        TFR     D,X             Déplace en X
REPT01  LDB     ,X              Entre valeur-en-cours
        BSR     PUTHEX          Affiche valeur-en-cours
        LDA     SPACE,PCR
        BSR     OUTCH           Affiche un espace
        BSR     GETVAL          Entre valeur-nouvelle
IF03    TSTA                    Si valeur-nouvelle n'est pas un point
        BLT     UNTL01
        BGT     ENDF03          Si ce n'est pas un retour
        STB     ,X              Stocke valeur-nouvelle dans emplacement-en-cours
ENDF03  LEAX    1,X             Incrémente emplacement-en-cours
        TFR     X,D             Affiche emplacement-en-cours
        BSR     DSPADD
        BRA     REPT
UNTL01  PULS    A,B,X,PC
```

Somme des parties

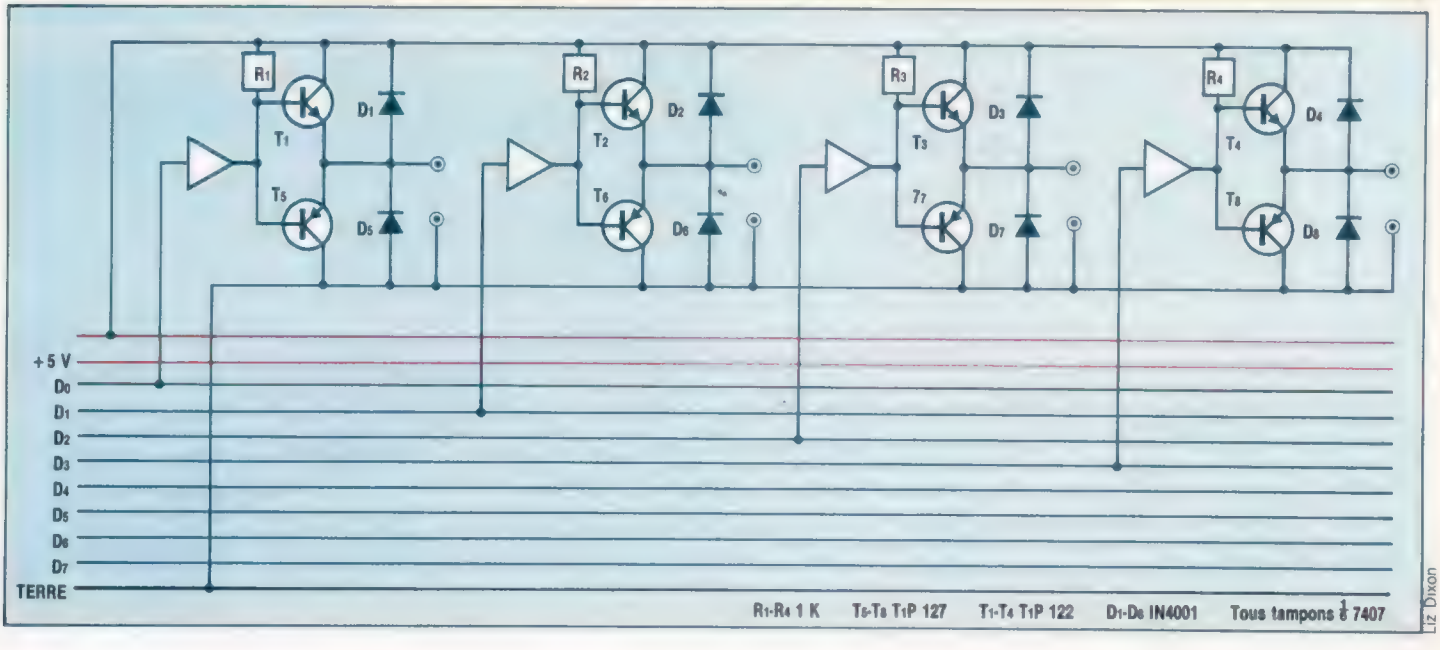
Nous avons mis au point un système électronique à tampon qui peut être utilisé avec le Commodore 64 pour contrôler et commander des périphériques extérieurs.

Le Commodore 64 dispose d'une configuration d'entrée/sortie qui lui permet de communiquer avec le monde extérieur au moyen d'un port utilisateur composé essentiellement de huit broches de données et d'une prise de terre. Ces huit broches sont branchées directement dans une adresse de mémoire particulière nommée registre de données (chaque broche correspond à un bit du registre). Une deuxième adresse, le registre de direction des données (RDD), commande la direction du débit des données sur chaque broche. Si une broche est mise en sortie (bit RDD = 1) une tension de +5 V est appliquée à la broche lorsque le bit correspondant est mis au niveau élevé (à 1). Si le bit de données est mis au niveau bas, une tension nulle est appliquée sur la broche correspondante. Bien que le courant fourni par le port utilisateur ne puisse actionner directement des dispositifs extérieurs, il peut servir à exciter un système de relais qui autorise la commande de systèmes à tension et/ou courant plus élevés.

Quand une broche est mise en mode entrée (bit RDD = 0), le fonctionnement est tout à fait différent. Là, le bit correspondant du registre de données est maintenu au niveau élevé, et ne passe au niveau bas que si la broche est mise à la terre. Ce fait peut être exploité pour contrôler des événements ayant lieu à l'extérieur du micro en connectant un côté d'un simple interrupteur à une broche de données et l'autre à la prise de terre du port utilisateur. Lorsque l'interrupteur est fermé, la broche de données est connectée à la prise de terre et le bit correspondant du registre passe du niveau élevé au niveau bas. Ce changement dans le registre de données peut être facilement détecté par un logiciel qui contrôle son état ; le déroulement du programme peut donc être modifié de l'extérieur.
Le système entier est conçu autour d'un bus commun à neuf lignes appropriées à ses besoins. Ce bus est relié à chaque dispositif au moyen d'un connecteur à douze broches.

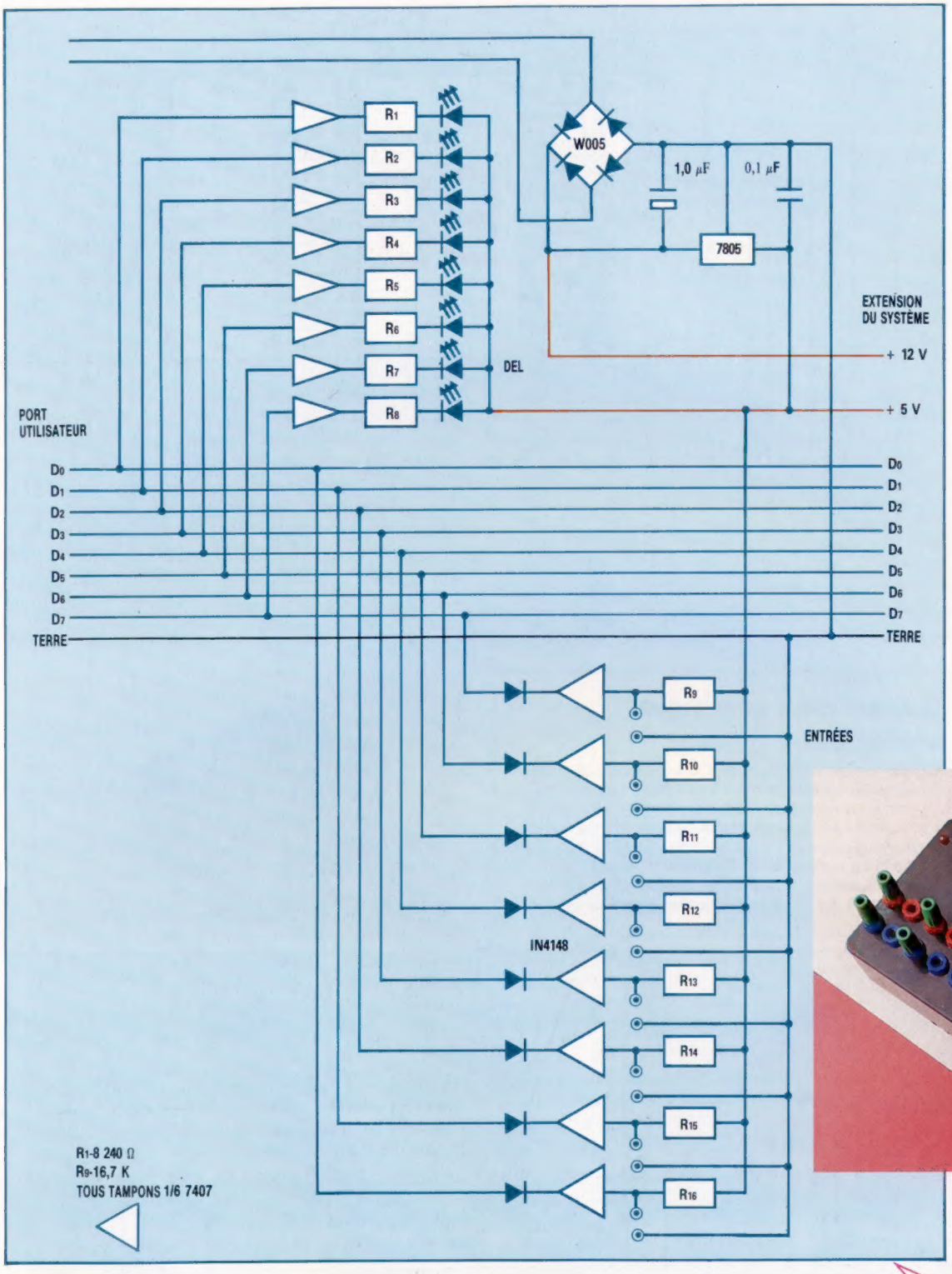
Le boîtier de sortie

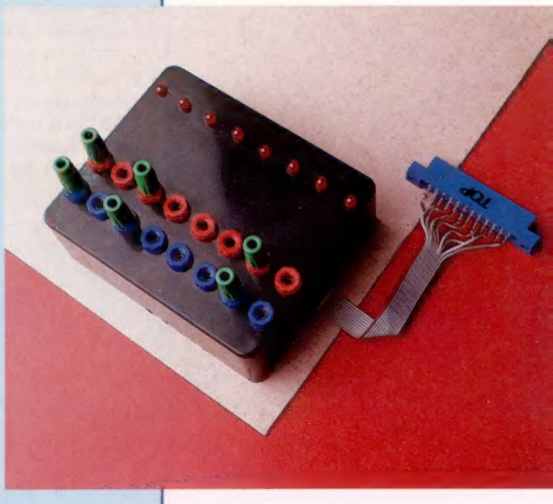
Le boîtier de sortie basse tension est branché au bus du système au moyen d'un connecteur mâle à douze voies. Le courant fourni par une broche de données élevées est de l'ordre de quelques milliampères — ce qui n'est pas suffisant pour actionner un dispositif comme un moteur électrique, mais l'est pour servir de courant de commutation *via* un transistor. Les lignes de données 0 à 3 sont utilisées par le boîtier basse tension. La mise au niveau élevé d'une de ces lignes entraîne la mise sous tension de la prise rouge correspondante de ce boîtier. Il est donc possible d'alimenter quatre dispositifs simultanément. Chaque ligne peut fournir jusqu'à 1 A, selon le transformateur utilisé.



Le boîtier tampon

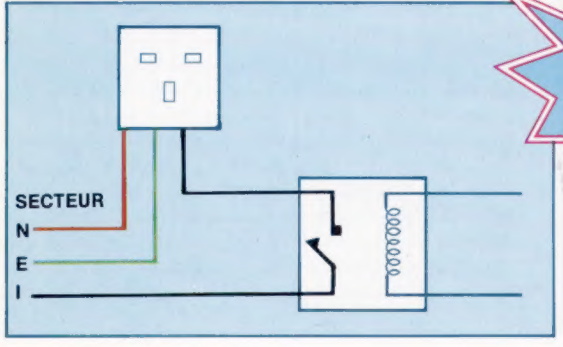
Le boîtier tampon est le premier et le plus important dispositif dans le système du port utilisateur. Les circuits protègent l'ordinateur contre toute tentative de tirer trop de courant d'une broche ou d'appliquer une tension d'entrée. Il accepte en plus une entrée CC ou CA provenant d'un transformateur, dans un éventail de 5 à 21 V, et en assure la régulation. L'entrée de tension est ajoutée au bus du système comme une paire supplémentaire de lignes en vue d'être utilisée par d'autres parties du système. Quand le boîtier tampon est connecté, il est possible de transmettre des informations au port utilisateur; les huit prises rouges correspondent aux huit lignes de données; les prises noires fournissent une prise de terre à chaque ligne de données. Pour donner une indication visible de l'état de chaque ligne de données, une série de huit DEL est montée sur le boîtier.





Le relais secteur

La tension du transformateur peut commander celle du secteur à l'aide d'un relais. Cette unité se branche au secteur et à l'une des quatre lignes du boîtier de sortie. La mise au niveau élevé de l'un des bits du registre de données a pour effet d'alimenter la prise de sortie correspondante par le transformateur, ce qui applique la tension du secteur sur la prise secteur à trois voies. Nous pouvons donc commander des dispositifs alimentés par le secteur à partir de l'ordinateur.

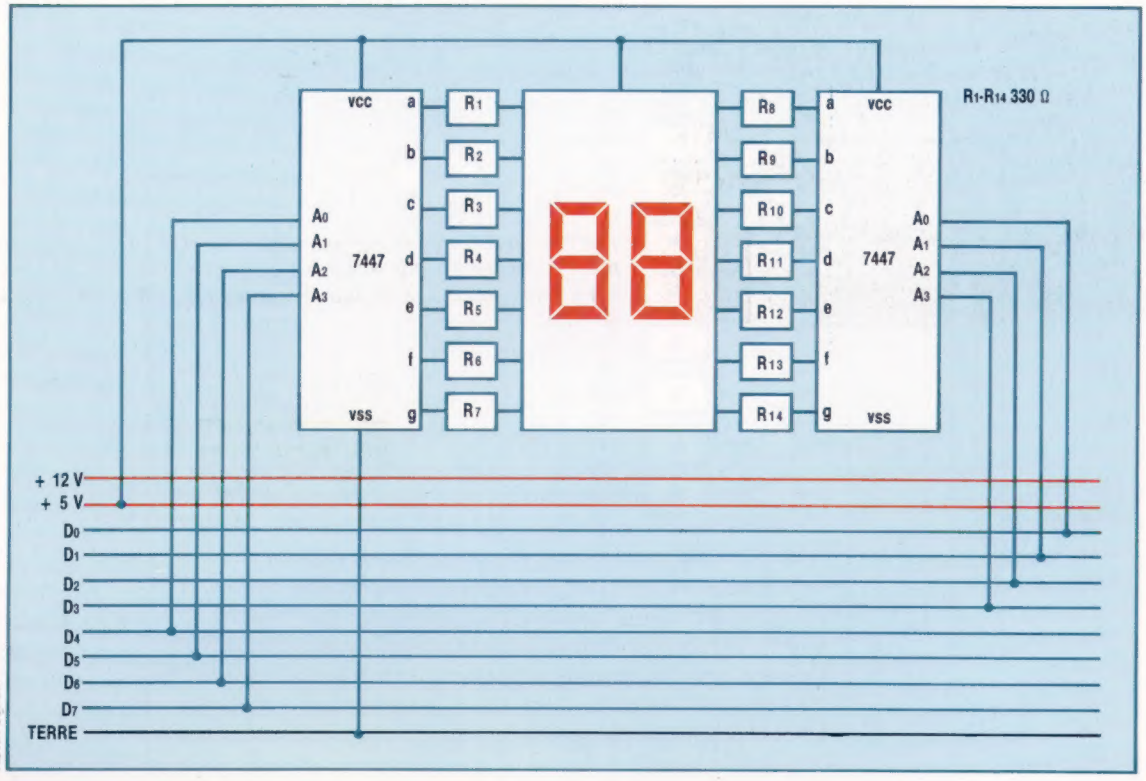


AVERTISSEMENT

- **Débranchez les sources de tension avant de commencer à travailler.**
- **Vérifiez toutes les connexions et tous les montages avec un multimètre.**
- **Évitez de provoquer des courts-circuits.**

Affichage à sept segments

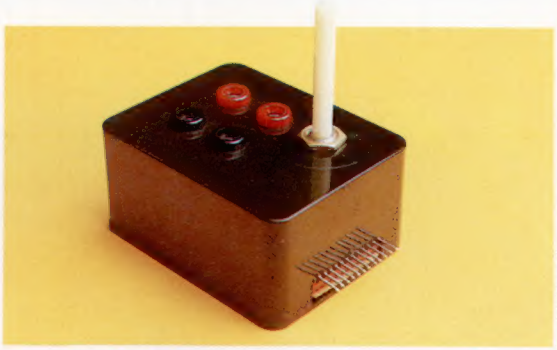
Un affichage à sept segments nécessite quatre entrées logiques pour afficher les seize chiffres hexadécimaux; nous pouvons donc piloter deux de ces affichages à l'aide des huit lignes de données disponibles sur le bus du système. Cette unité affichera donc le contenu du registre de données du port utilisateur comme une paire de chiffres hex.



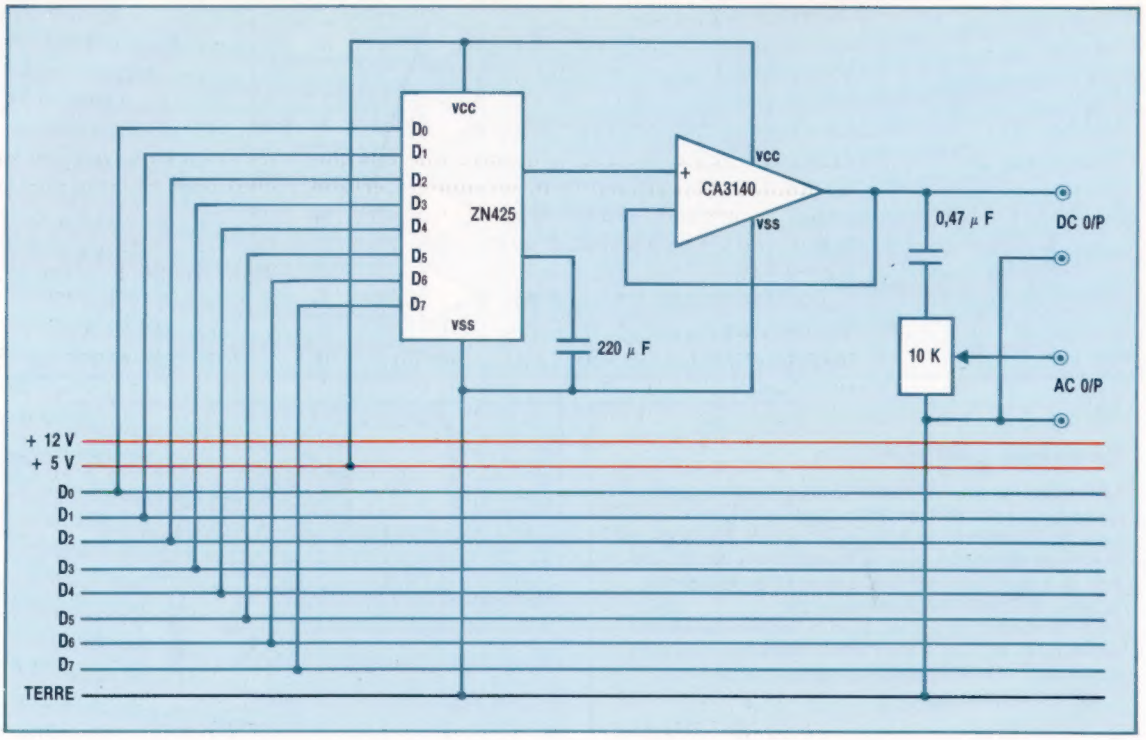
Kevin Jones

Convertisseur numérique analogique

Le convertisseur N/A convertit une valeur du registre de données comprises entre 0 et 255 en une tension. La sortie provenant du boîtier est d'environ 600 W; c'est insuffisant pour alimenter des lampes ou des moteurs directement, mais peut être amplifié pour y parvenir. Le convertisseur N/A peut aussi servir à générer des sons, soit directement au moyen d'un casque, soit par l'intermédiaire d'un amplificateur audio.



Ian McKinnell



Page manquante
(publicité)

Page manquante
(publicité)